The University of St. Andrews 17 April 1997

CS3015: Senior Honours Project

A Computer Aided Software Engineering Tool

-Project Report-

Martin Bramley

Abstract

The aim of writing this project was to design and implement a graphical class hierarchy designer for the object-oriented programming language C++. The plan covered a survey of similar tools, the specifications for the implementation and the outline implementation plans. The report summarises the project and its implementation.

Overall the project was successful resulting in a working class browser which was editable enabling users to load, save and generate C++ class hierarchies.

Declaration

Contents

Part 1

Project Plan

The University of St Andrews 15th November, 1996

CS3015: Senior Honours Project

Resworb

A C.A.S.E Tool

-Project Plan-

Martin Bramley

Contents

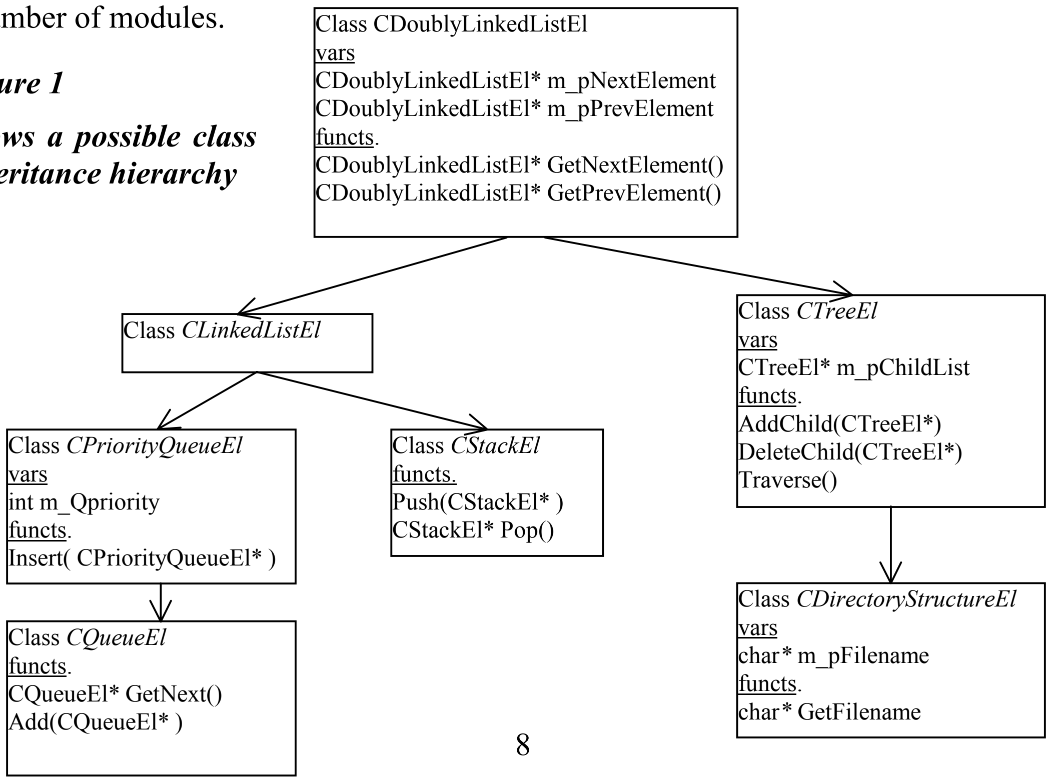
1.1 Problem Definition

Class browsers are useful because they allow the user to see how classes are interrelated (by inheritance, containment or reference) with one another. This enables the user to see how classes interact with one another. There are many class browsers available for a wide range of object oriented languages. The fundamental drawback of all of these browsers is that they may only be used after the code has been compiled. This is because most class browsers are actually packaged within a programming environment, for example Borland C++ and Microsoft Developer Studio, which contain its own compiler such as Borland C++'s BCC.EXE. It is within the compilation process that the browser data is generated. Would it not be better to browse and even edit the class hierarchy before any code was compiled by separating the generation of this browser data from the compilation process? This would allow the browsing of class data of incomplete programs.

A system like this would facilitate the design and implementation of other programming projects by enabling the programmer to visually interact with their code on a class basis. This would allow the programmer to construct a class hierarchy in similar method to that of constructing a cloud diagram. For example the following diagram (fig. 1) would be easier to understand than a lot of code separated into a number of modules.

Figure 1

Shows a possible class inheritance hierarchy



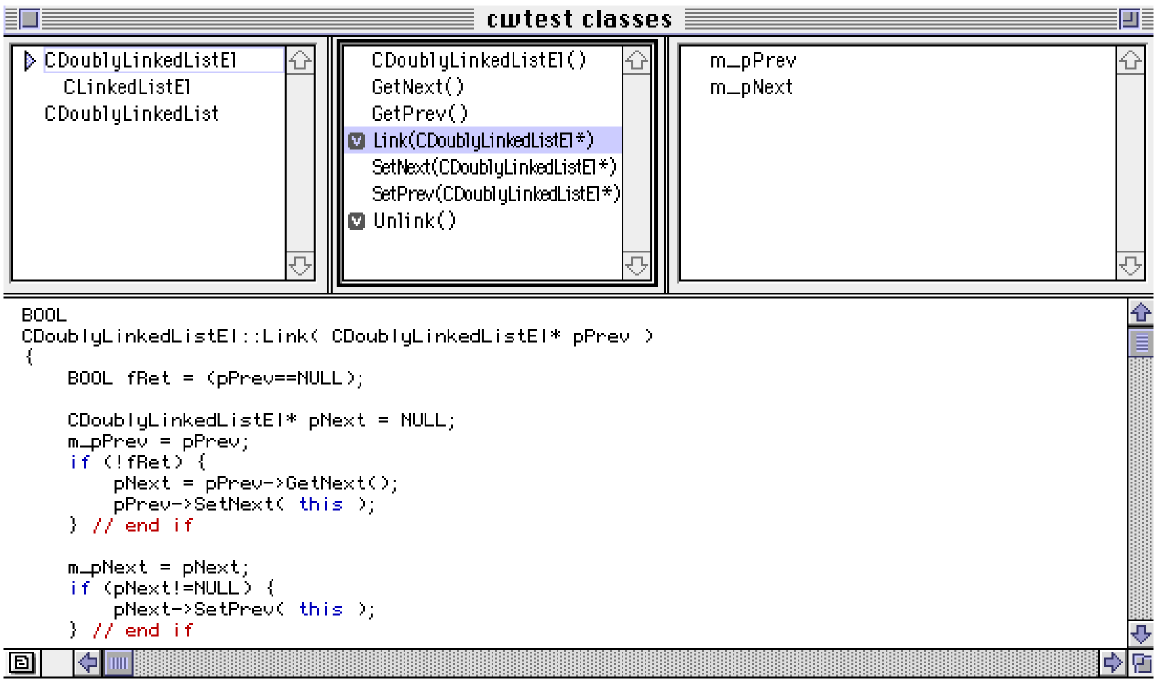
1.2 Context Survey

1.2.1 Reviewed Software

After reviewing a few software packages which contain class browsers it became apparent that the majority of them were post-compilation browsers which made the class browsers non-editable. After extensive research on the Internet it was concluded that there are two browsers which work before compilation. These were Relational Rose and Visual Thought. Unfortunately, demonstration packages were not available to be evaluated. The post compilation packages surveyed were MetroWerks CodeWarrior, Semantic Think C++, Borland C++ and Microsoft Developer Studio.

1.2.1.1 MetroWerks CodeWarrior

The class browser with this package used four text windows (see fig. 2 below). The top left window contains the class hierarchy which seems a little on the small side considering there may be many classes within an application. Within this window inheritance was indicated by indenting the child class in from its parent. This is not very clear if there is a large inheritance tree as it is too easy to make mistakes in interpretation. The middle window contained a listing of all the member functions declared within the selected class. This may seem fairly acceptable, but would it not be better to show all the functions available to the class, including the inherited functions which could be shown in a different colour? The bottom window contained the code for the selected member function. This is fine, although as it is post-compilation browser the code is not editable. The top right window contains the declared member variables within the currently selected class. Overall, this browser served its purpose, although it is not always necessary to see the code for the associated member functions, and improvements to the display of the class hierarchy would be a bonus.



```
CDoublyLinkedListEl
  CLinkedListEl
CDoublyLinkedList
```

```
CDoublyLinkedListEl()
GetNext()
GetPrev()
Link(CDoublyLinkedListEl*)
SetNext(CDoublyLinkedListEl*)
SetPrev(CDoublyLinkedListEl*)
Unlink()
```

```
m_pPrev
m_pNext
```

```
BOOL
CDoublyLinkedListEl::Link( CDoublyLinkedListEl* pPrev )
{
    BOOL fRet = (pPrev==NULL);

    CDoublyLinkedListEl* pNext = NULL;
    m_pPrev = pPrev;
    if (!fRet) {
        pNext = pPrev->GetNext();
        pPrev->SetNext( this );
    } // end if

    m_pNext = pNext;
    if (pNext!=NULL) {
        pNext->SetPrev( this );
    } // end if
```

Figure 2 MetroWerks CodeWarrior

1.2.1.2 Semantic Think C

Semantics Think C uses a graphical class browser (see fig. 3). This means it is far easier to interpret the inheritance trees, although using arrows would help clarify things. Unfortunately the user has no way of manipulating the layout of the class hierarchies. Another drawback is that it is not possible to see the member functions of a class unless the mouse button was pressed over the selected class. This is a hindrance as it is often useful to see the member functions of a number of classes at the same time. To see the code associated with the member functions all that is required is to release the mouse button over the member functions whose code is to be viewed. The code was displayed in a separate window. Again the source code is not editable. Overall this browser looks better than CodeWarrior's, but the user-interface leaves a lot to be desired.

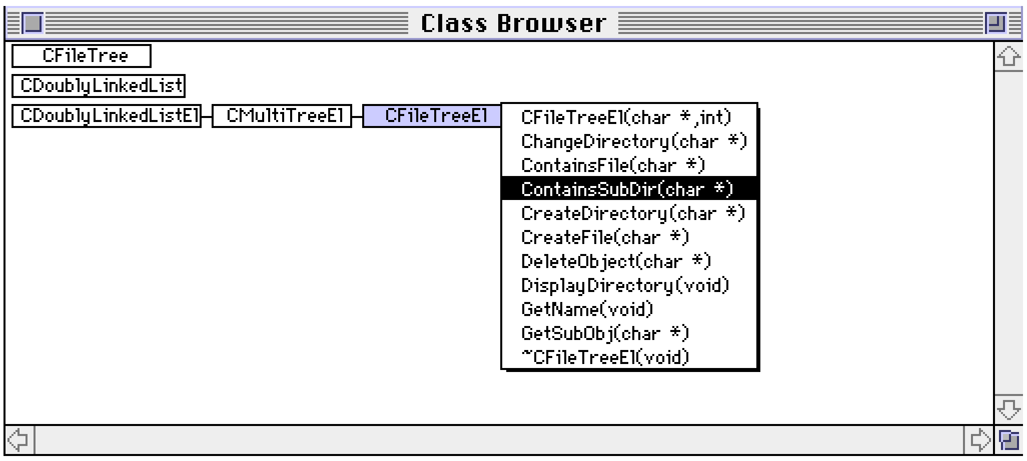


Figure 3 Semantic Think C

1.2.1.3 Borland Visual C++

As you can see in figure 4, this package's object browser is at first glance surprisingly similar to that of Semantic Think C. However, there are some improvements which are mostly in the user interface. To view the functions available to a class a double click on the class brings up another window (see Fig. 5) which contains all the functions available to the class and shows where they originate. To view the code within a function a double click with the mouse on the function brings up the code within a text viewer. Here the function code may be edited but any changes to the class structure may not be viewed until the entire application is rebuilt.

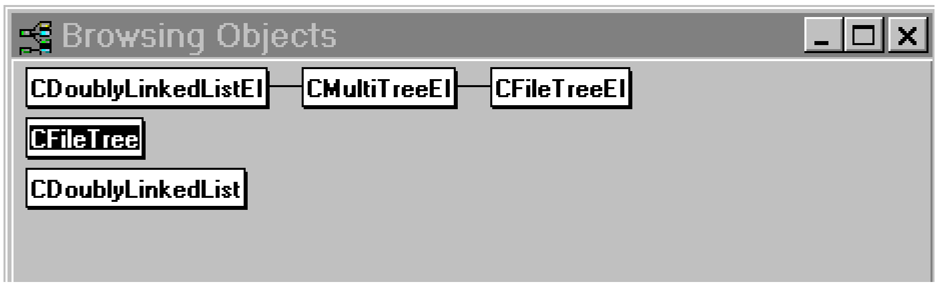
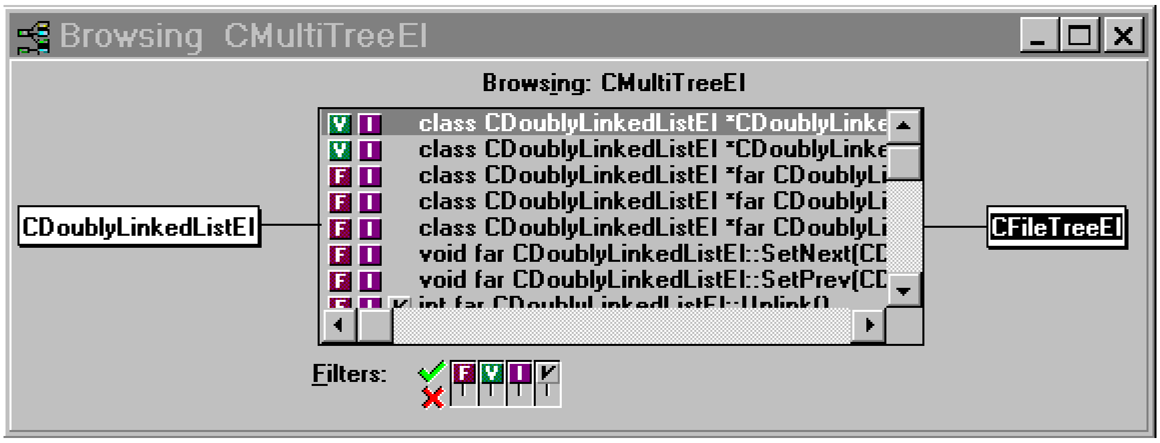


Figure 4 Borland Visual C++ (Object Browser)

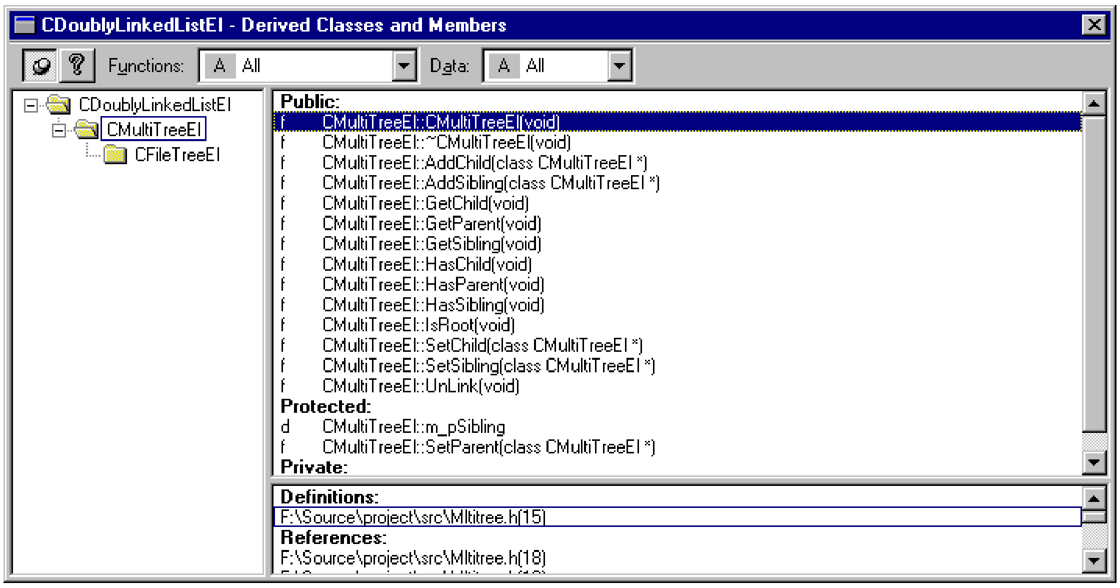


```
V I   class CDoublyLinkedListEl *CDoublyLinke
V I   class CDoublyLinkedListEl *CDoublyLinke
F I   class CDoublyLinkedListEl *far CDoublyLi
F I   class CDoublyLinkedListEl *far CDoublyLi
F I   class CDoublyLinkedListEl *far CDoublyLi
F I   void far CDoublyLinkedListEl::SetNext(CD
F I   void far CDoublyLinkedListEl::SetPrev(CD
```

Figure 5 Borland Visual C++ (Class Browser)

1.2.1.4 Microsoft Developer Studio

This browser is by far the best of those reviewed . The browser’s inheritance tree is the simplest to understand and may be collapsed or expanded by the user. The top window contains the member functions and member variables of the class. The level of detail shown in this window may be changed by using the combo-boxes in the toolbar. It is, however, the bottom window which makes this browser more usable than any of the others. The window contains references to where the selected member function is defined and called. This is useful because when the user double clicks on one of the references, the function containing the reference is brought up in an editable window. The only drawbacks with this browser is that you may only browse once the application has been successfully compiled, therefore any changes to a class structure may only be seen once the application has been rebuilt, and there is a lot of extra disk space required to contain the class data.



```
Public:
f       CMultiTreeEl::CMultiTreeEl(void)
f       CMultiTreeEl::~CMultiTreeEl(void)
f       CMultiTreeEl::AddChild(class CMultiTreeEl *)
f       CMultiTreeEl::AddSibling(class CMultiTreeEl *)
f       CMultiTreeEl::GetChild(void)
f       CMultiTreeEl::GetParent(void)
f       CMultiTreeEl::GetSibling(void)
f       CMultiTreeEl::HasChild(void)
f       CMultiTreeEl::HasParent(void)
f       CMultiTreeEl::HasSibling(void)
f       CMultiTreeEl::IsRoot(void)
f       CMultiTreeEl::SetChild(class CMultiTreeEl *)
f       CMultiTreeEl::SetSibling(class CMultiTreeEl *)
f       CMultiTreeEl::UnLink(void)
Protected:
d       CMultiTreeEl::m_pSibling
f       CMultiTreeEl::SetParent(class CMultiTreeEl *)
Private:
```

```
Definitions:
F:\Source\project\src\Mltitree.h(15)
References:
F:\Source\project\src\Mltitree.h(18)
```

Figure 6 Microsoft Developer Studio

1.2.1.5 Editres

This is not a class browser but a resource viewer for motif applications. The user interface is very good as it allows the user to see how the resources fit together to create the finished program, and it leaves no room for confusion. The only drawback for this package is that different levels of resource inheritance are not contractible (i.e. the entire resource tree is always shown). Although this is not a problem it may generate a resource tree which is cumbersome to view. Another drawback is the fact that the tree is not user manipulable.

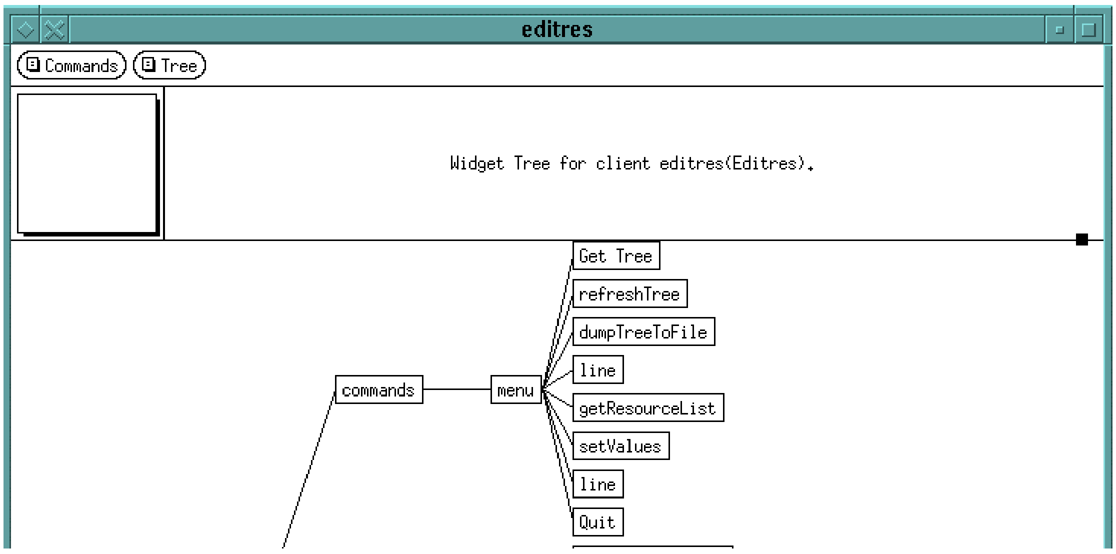


Figure 7. Editres

1.2.1.6 Conclusion

The class browsers reviewed provide the ability to browse class hierarchies to a greater or lesser extent. CodeWarrior’s browser was an interesting solution as it managed to convey a lot of information without the use of graphics. The browser within Microsoft Developer Studio is by far the most usable, although this seems to fall some what short of fulfilling the problem, primarily because of it post-compilation nature which restricts the easy manipulation of class structures.

1.2.2 Background Research

With the project aiming to facilitate the implementation of C++ classes it seemed appropriate to examine the techniques available in the design and implementation of object-oriented programs. The first technique was designed by Coad and Yourdon, who broke the problem down into three distinct stages. The first stage is the analysis of the problem. The implementer starts at this stage with a broad overview of the problem in terms of objects, then the implementer refines the objects by adding a little detail and finally iterates through this stage again continually adding details until a simple hierarchy which solves the problem is derived. This may be augmented by a real world representation of the model to help capture the problem and understand it. The second stage of the technique is known as the design stage. Within this stage the solution from the first stage is taken and the computational restraints of the model are addressed and solved. The last stage is the implementation of the refined problem with the constraints applied by the second stage.

The second technique is Booch method. This technique addresses the analysis and design of object oriented systems. It may be simplified and be broken down into three stages. The first stage is the analysis of the problem. At this stage the problem is analysed, specified and defined; by the end of this stage the implementer should know all the *whats* of the problem. This is achieved by generating uses_cases. A uses_case specifies a relationship between two entities; for example a footballer owns football boots. Once all these uses_cases have been generated they are split down using interaction diagrams helping express the functionality captured. For example :-

```
Footballer scores goal
        footballer selects angle
        footballer kicks ball
                Ball moves
                Ball Stops
                        Does Goal contain Ball?
```

The differing levels of indentation signify different entities. It is now obvious that these real world entities which have interactions may also be modelled as classes within an object-oriented model.

The second stage of the technique is used to specify which entities must exist if another object exists, and the last stage is to implement the classes.

1.2.2.1 Conclusion

Both techniques have their advantages and disadvantages. The Booch technique unfortunately has a very large set of symbols which may confuse matters, but by having this large set of symbols it is possible to generate the ideal abstraction and then to directly translate the symbols within the abstraction to working code. The bonus of the first technique must be its simplicity. There is a adaptation of this system which closely allows the code to evolve more than be designed. This is achieved by implementing a very basic system and then extending it as the problem is refined.

1.3. Problem Specification

1.3.1 Functional Requirements

The user of the system will want to perform four key operations - create a new project, load a project, save the current project and edit the current project.

1.3.1.1 The Menus

There will be a single menu with in the application labelled *File*. This file menu will contain a *Create New* option, *Load* Option, *Save* Option possibly a *Save As* Option and a *Quit* option.

Creation

The first operation a user will require is to create a new project. This may be achieved by the user selecting the *Create New* menu option. Once selected the user will be prompted for a project name and location; this will be achieved by using a graphical file window. After an acceptable project name has been entered a project directory of that name will created and it is within this directory that all the files generated by the application will be stored. All projects will contain a default class from which all other classes must inherit. This class will contain some virtual debugging and printing facilities, such as a function which will return the type of the object as a string. There may be a little performance hit in doing this, but, as specified in the Non-functional requirements, the user should be able to remove this base class themselves.

Loading a Project

To load a project the user must select the *Load* option from the *file* menu. The user will then be asked to select the project directory they wish to load through a graphical file window.

Saving a Project

To save a project the user must select either *Save* or *Save As* from the *File* menu. *Save* will save to the same project directory it was loaded from and *Save As* will save to a new project directory which the user must enter through a graphical file window. Within the directory a project is saved, two files will be generated for each class (a C++ header file and a C++ code file) and additionally a single binary file which will contain information about the positions of classes on the screen and other display options. The user may edit the generated C++ files, although it will not be possible for the user to add or remove classes as they will have entries in the binary file. The user may add or remove member variables or member functions to the classes as long as the order they occur in the header file is the same as the order the occur in, in the code file.

1.3.1.2 Editing a Project

There are two main types of editing a user may wish to perform. The first is editing the layout of the class inheritance tree. This is simply the user moving the classes around the graphical canvas and the computer updating the links between the classes. A single class may be moved or a selection / group of classes may be moved together. The user may also wish to expand or collapse a branch of the inheritance tree. This will be implemented entirely by the use of the mouse. The second type of editing is the editing of individual classes.

The user may edit the user-classes by the use of context sensitive menus which will be bound to the right hand mouse button. The first thing a user will wish to do when editing a project is to create a class. This will be achieved by the user pressing the right mouse button over the intended parent's class name This will cause a pop-up menu to appear where one of the options will be to create a child class. A new class will then be created on the screen which will have a default name and description which the user must modify.

Once the user has created a class the user may wish to add member variables or member functions. This will be achieved by the user pressing the right mouse button over the class name and then selecting *add variable* or *add function* from the pop-up menu which will appear. If a variable is to be added a dialogue box will be displayed asking for the user to enter the variable type and name. If a function is to be added an editable text window will be displayed allowing the user to enter the function and with it a comment or description.

Next the user may wish to edit or delete member variable or member functions from a class. This may be achieved by the user selecting the member to be removed within the class with the right mouse button and then selecting delete member from the pop-up menu.

The user may wish to do two forms of deletion. The first is to delete a branch of the class hierarchy. To do this the user should select the class at the top of the branch to be deleted and then indicate that it is to be deleted. This again will be by using the right hand mouse button to bring up a pop-up menu but the option to delete will require keyboard confirmation. This should not be made too easy as the process will be irreversible and all sub-classes will be deleted at the same time. The second form of deletion which will be available is where the contents of the deleted class are copied down into it children and the deleted class's children are linked to the deleted class's parent. Again, this will be achieved by using the pop-up menu bound to the right mouse button.

The user may also wish to move an entire class to a different point within the inheritance tree. This will be achievable in two ways; the first by making the links between classes drag & drop-able themselves; and the second being where the user can select a branch of the tree and cut it out and then paste it within the tree at another point. A move like this is potentially very dangerous as the class may depend on its previous parent and the application will not do any checking for inconsistencies.

One of the most powerful tools in object oriented programming is the idea of multiple inheritance. This is obviously a required feature. So far there has been a heavy emphasis on hierarchy trees, whilst really it should lie on hierarchy graphs. This emphasis is primarily to reduce the potential confusion of the user. A possible solution would be to add a function (such as *add_a_parent*) to the pop-up menu, which will make the following class clicked become the parent of the class from which the function was called.

So far inheritance is the only form of class relationship covered. The other forms of relationship (containment and pointer) have not been addressed. This will be shown graphically using different coloured arrows for each relationship. This may have to be discarded if it makes the inheritance tree too convoluted.

1.3.1.3 Other Operations

Other operations the user may wish to perform could be changing the size of the graphics pad or using snap to grid. Operation such as these will be accessible using the menus at the top of the window. They should be saved in an application defaults file.

1.3.2 The Graphical User Interface

As defined the system is aimed at advanced users. The G.U.I (Graphical User Interface) will be designed for use by someone who is competent with graphical programs. The interface should be as free of clutter (unnecessary button bars etc.) as possible. This will be achieved by using context sensitive pop-up menus. There will be a set of menus at the top of the window for application and project manipulation tools (i.e. saving and loading). In general the left hand mouse button will be used to manipulate the position of the classes and links on the screen whilst the right hand mouse button will be used to display the context sensitive pop-up menus which will contain options for editing the class. When the right hand mouse button is pressed over a class name the pop-up menu options will be *create a child class*, *add member function*, *add member variable*, *edit class name*, *edit class description*, *cut*, *delete branch* and *delete class*. If the right hand mouse button is pressed over a member variable or function the pop-up menu options will be *edit member*, *delete member*, *make private*, *make public* and *make protected*.

There is a program available to view the resources of motif applications called Editres (See context survey). This program has a similar graphical user interface to that which it is hoped will be achieved herein.

1.3.3 Non-Functional Requirements

1.3.3.1 The User

The user of this application is expected to be relatively experienced in the use of the object oriented programming language C++. It is expected that the user will be confident with Graphical User Interfaces. The user should also be able to grasp the basic concepts of the application with ease and with the aid of the instruction manual they should be able to master the application.

The program will not be multi-user and it will not support having more than one project open at a time, although it will allow multiple instances of itself to be run at a time.

1.3.3.2 Documentation

There will be two forms of documentation supplied with the finished system - one for the user and one for the maintainer. The users documentation will assume an elementary knowledge of C++ and will emphasis the concepts behind object-oriented programming, whilst explaining how the application may be used. There may also be some on-line help.

The maintenance document will contain a complete listing of the heavily commented source code; a document containing the concepts applied to develop the application; where specific modules may be found and what they do; and a list of possible enhancements with suggestions on how they may be achieved.

1.3.3.3 Hardware

The application will be implemented using motif or motif++ and C++, and thus it should fairly portable, although it will be specifically designed to run on the computers in the Philip Lee Laboratory, which are Dec Alpha 300LX Workstations with 32 megabytes of Internal RAM and 512 megabytes of internal disk space. The processor is a 125MHz RISC CPU.

1.3.3.4 Performance Constraints

The application will be expected to run at a speed which will feel comfortable to the user and so the user should not be left waiting. In general the system will respond in less than three seconds unless the computer is busy on another task. Exceptions to this will be starting, loading and saving, processes in which large amounts of data is being manipulated.

The finished application will only be capable of having one project open at a time. In addition the graphics pad may be of finite size.

1.3.3.5 Error Handling and Extreme Conditions

There are three classes of errors - environmental errors, application errors and user errors. Environmental errors are errors which occur due to no fault of the application or the user, for example poorly set up environment paths, lack of space on the file system or a bad network connection. In general it will be hoped that the application will catch most of these errors and report back to the user. However, this can not be guaranteed due to the wide range of possible errors.

Application errors are errors in the application itself due to poor implementation. In implementing the project a very defensive stance will be taken with two levels of assertion catching; debug asserts and release asserts. The implementer should use debug asserts whenever a decision within the program is made. When a debug assert fails the user should be alerted to the nature of the assertion failure and where it happened. These asserts should be changed to release asserts in the release version. The second type of assertion is a release assertion. These assertions should be used when anything potentially disastrous may occur. When a release assertion fails the user will be told that an internal error has occurred and then the application will try to save the current project.

User errors are errors generated when the user attempts to do something illegal. The application should not allow this to happen and when a user attempts this the user should be alerted of the error.

1.3.3.6 System Modifications

Areas of the application which may be improved upon at a later date are :-

- An interface to an R.C.S.
- Support for other languages apart from C++ (possibly Napier, Small-talk or Eiffel)
- Extending it to a fully integrated programming environment with a debugging tool
- Extending it to have multiple projects open at a time with cut and paste facilities between projects.
- Adding some error checking within the modules (i.e. checking the user declares a variable before using it).

1.3.3.7 Deliverables

There are two sets of deliverables. The first will be a project plan and the second being a project report.

The plan will define the problem, survey existing products and relevant back-ground material, specify the problem in terms of what requirements it should fulfil, a document covering how the project should be implemented and finally a project monitoring sheet specifying the estimated timings of the project.

The report will outline the project, contain an updated version of the plan, summarise the project, outline the testing done, a user manual, a maintenance manual and a status report.

The plan is due in at 4 p.m. on 15th November 1996 and the report is due in on 4 p.m. on the 19th of April 1997

In addition the project will be presented to staff and students at a later date.

1.4. Implementation and Testing plans

1.4.1 Implementation

The implementation of the program must run on the DEC Alpha, in a UNIX windowing environment (See Non-functional Requirements). The graphical user interface will be implemented using an object oriented extension of the standard motif widget set called wxWindows. There are a number of benefits of using this system instead of TCL / TK, which is another possibility. The primary benefit is that the implementer has more power over how the interface interacts with the underlying program, whereas TCL / TK is restrictive in the way that it communicates with the underlying code, since it has to be achieved entirely by string manipulation. Another benefit is that wxWindows code may be precompiled while TCL / TK code is interpreted at run-time which make large TCL / TK applications sluggish. Thirdly, wxWindows is very portable as it may use any widget library and not just motif, so in theory it should be possible to compile and run the system under Linux and Microsoft Windows 95. The last benefit of using the motif library is that the user may modify the generated resource files to add their own menus and functions.

The underlying language which will be used to implement the application will be C++, because the nature of the application being implemented is very object based and I feel that an object oriented language will prove to make its implementation easier.

To enable ease of future maintenance an adaptation of the Hungarian notation will be used throughout the applications code. The drawback with Hungarian notation is that it names variables with regard to what they are, whereas most programmers would prefer to name variables by what they do. The adaptation uses an abbreviated form of Hungarian notation to let the user know what he is dealing with and the rest of the name will be used to describe the purpose of the variable.

Due to the nature of object-oriented design it is possible to separate the program into distinct modules. However, because the user is generating classes and the application will contain classes, there is a chance this report may become unclear. Anything the user generates, i.e. classes and code, will be referred to as user-classes and user-code, and anything which is part of the program's implementation will be referred to as program-classes and program-code.

1.4.2 The user interface

Within Editres it seems (although this is not certain) that each resource is actually a button. The implementation of this project will use a similar method, where each user-class is actually a window which will contain a number of buttons. These buttons will act as bindings to the context sensitive menus which will occur when the right mouse button is pressed.

Each user-class will contain at least one button which will be the class-name. This button will contain the binding which controls the context sensitive pop-up menu of the user-class. The other buttons within these windows will be for either the user-member-variables or the user-member-functions. These buttons will contain bindings to a different set of pop-up menus which will enable the user to change the access level, delete or edit the member.

1.4.3 The Modules

The application will contain four main modules and one module which will contain miscellaneous utilities and class. These modules will fit together in the following way:-

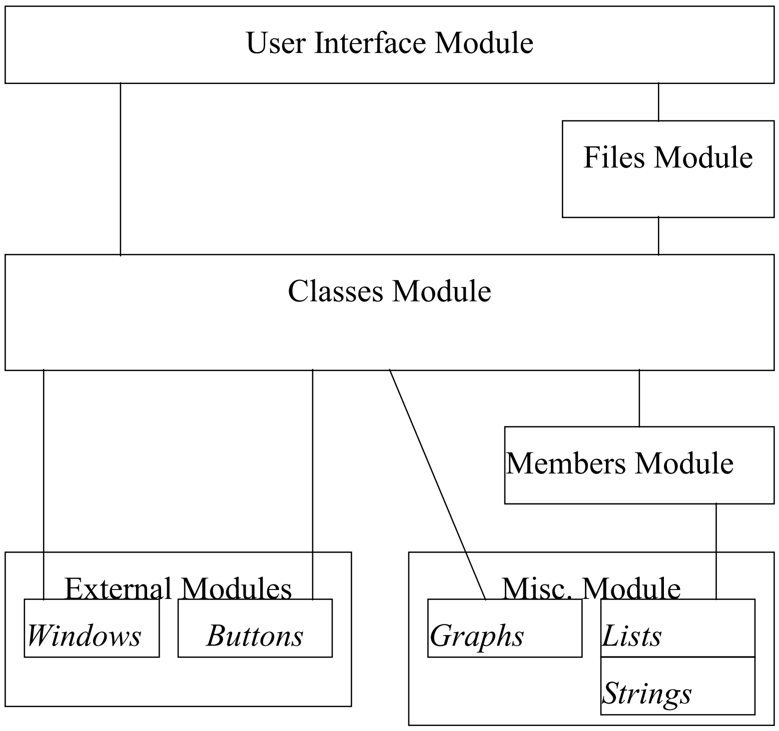


Figure 8. How the modules connect.

The modules will be implemented in the following order. A basic G.U.I will be implemented to allow testing during development, then the miscellaneous and classes module will be developed, and finally the files module and the interface code between the classes module and the user interface module will be implemented.

1.4.3.1 User Interface Module

This module will contain the code which will generate the bare-bones of the graphical user interface. The module will create the main window, the menus and the canvas which will contain the users-inheritance tree. All menu actions which occur within the *File* menu will make calls to the file access module. All actions which occur within the canvas will be dealt with by the classes module.

1.4.3.2 File access module

This module will contain a number of functions which may be called externally. These functions will do a number of checks and then call one of the following two functions (a load function and a save function). The load function will load the project name given as a parameter and if the project name does not exist then it will create the project. The load function will firstly read the binary file within the project directory, which will contain information about what user-classes exist and where they are to be positioned on the screen. It will also contain information on how the user-class hierarchy should be traversed to get the class relationships corrects. The function will then traverse the generated users-class hierarchy calling the member function *load_class*.

The save function will save the users project to the specified directory. This function will actually traverse the users-class tree, firstly saving position data to the binary file and then calling the member function *save_class*.

1.4.3.3 Classes Module

This module will contain the implementation of the program-class which contains the user-classes. The program-class will inherit from a labelled cyclic graph (implemented in the misc. module) to enable multiple inheritance. It will also inherit from two wxWindows classes which will contain all the graphical information (i.e. position and size)

This program-class will contain four member variables and a number of member functions. The structure of this program class maybe something like :-

```
class CDisplayClass : private CWeightedCyclicGraph{
private:
        char*                   m_pClassName
        char                    m_ClassDescription[256]
        CMemberFunction*        m_pFunctions            // a Double linked list of functions
        CMemberVariables*       m_pVaraibles            // a Double linked list of variables
public:
        AddVariable
        AddFunction
        MakeChild
        AddParent
        UnLinkParent
        EditName
        EditDescription
        LoadClass
        SaveClass
}
```

Where the public functions, except *LoadClass* and *SaveClass,* are the functions available to the context sensitive pop-up menu available through the right hand mouse button. The *AddVariable*, *AddFunction* and *MakeChild* functions will create default instances of the sub-object which will be added to the lists for the user to edit. The *LoadClass* and *SaveClass* functions will be called by the file access module.

The private variables *m_pFunctions* and *m_pVariables* are going to be pointers to a doubly linked lists of program-objects, wherein each object in the list is going to be either a user-member-variable or a user-member-function.

1.4.3.4 Members Module

This module will contain the classes *CMemberFunction* and *CMemberVariable* which are used by the classes module. The classes will contain character data for the members it contains and functions to access such data. The classes will inherit from a class called *CMember* which in turn inherit from a list class defined in the misc. module. The *CMember* class will contain labels to specify what member access control the member has.

1.4.3.5 Miscellaneous Module

This module will contain service classes. It will contain a class *lists* to be used by the members module, a class *graphs* to be used by the classes module and a class strings which will be used in a number of places. It will be hoped that some of these may not have to be implemented.

1.4.4 Testing

1.4.4.1 Of the Classes

At each stage of implementation of the classes each one should be able to be tested rigorously for its correctness. Every public function within every class must be tested for its correctness. This may be achieved by writing simple test harness programs which access the classes being tested through the member functions and monitor the return results. It is impossible to test the classes with all data, so the implementer should examine the class and select test data which may be border line cases.

1.4.4.2 Of the Program

Once all the classes have been extensively tested and accepted, the testing of the entire program may begin. This will be tested in five stages. The first stage is when the programmer tests the program entering correct data. The next stage is again tested by the programmer but this time using erroneous data. The rest of the test stages are by potential users. Firstly they will test the program with valid test data and then again using the same erroneous data. Lastly the test users will be able to use the system on their own. If possible, beta versions of the program may be made available for testing. This will be used to allow potential users to comment on or suggest improvements for the program.

1.5. Commentary

1.5.1 Project Log

Tues. Sept. 24th

Undertook the project.

Wed. Sept. 25th-28th

Brainstormed on ideas and possibilities.

Fri. Oct. 4th

Searched Internet for related articles and similar programs.

Mon. Oct. 7th

Searched for related texts in library (nothing found).

Tues. Oct. 8th

Sent away for demonstration software which seem to do similar.

Thurs. Oct. 10th

Examined news-groups for useful information.

Fri. Oct. 11th-14th

Wrote simple non-platform specific C++ program and used this to review other class browsers.

Wed. Oct. 16th

Designed preliminary graphical user interface.

Fri. Oct. 18th

Returned to library and continued looking for relevant articles or books (nothing found).

Sun. Oct. 20th

Searched B.I.D.S. (still no articles found).

Tues. Oct. 22nd-25th

Got motif++ installed and started to examine it.

Mon. Oct. 28th

Revised user interface.

Thurs. Oct. 31st

Brainstormed over possible implementation styles and file access methods

Fri. Nov. 1st

Started collating information and drafted project plan. Found possible references in library and on the W.W.W.

Sat. Nov. 2nd-10th

Expanded project plan. Continued to experiment with motif++.

1.6. Project Monitoring

The project timetable is shown on the following page. It gives approximate starting and completion dates for different components within the program. Delays are inevitable, as with all large projects, although allowance has been made for these in the timetable. So far the project plan has been completed on schedule.

1.6.1 Dealing with Delays

As previously mentioned, delays are inevitable especially if modules keep failing their acceptance test. Each stage has been given 25% more time than originally planned and it is hoped to have the project completed a few weeks before the dead-line to allow debugging and fine-tuning.

1.6.2 Project Monitoring Sheet

Week No	8	9	10	11	12	Vacation	revision	Exams	1	Essay	3	4	5	6	7	Vacation	8
Week Ending	15-Nov.	22-Nov.	29-Nov.	06-Dec.	13-Dec.	08-Jan.	15-Jan	29-Jan	07-Feb	14-Feb	21-Feb	28-Feb	07-Mar	14-Mar	21-Mar	12-Apr	19-Apr
Internal Specifications of Modules		↔															
Implement List Module			↔														
Test List			↔														
Implement GUI			↔	↔													
Test GUI				↔													
Implement other Modules				↔	↔	↔											
Test other Modules					↔	↔			↔								
Refining Stage									↔								
Retest Modules											↔						
Test Program												↔					
Document Modules						↔											
Debug Program													↔				
Re-test with beta testers														↔			
User Documentation															↔		
Debug														↔	↔		
Complete Program																↔	↔

Figure 9 Proposed Project time schedule

1.7. Bibliography

1.7.1 Programs Used

CodeWarrior
MetroWerks.

Think C
Semantic.

Visual C++ Version 4.02
Borland International Incorporated 1993.

Developer Studio
Microsoft Corporation, 1994-1995.

Editres Version 1.1
Part of standard X11 distribution.

Motif Version 1.2
Open Systems Foundation.

wxWindows Version 1.66b
Julian Smart , Artificial Inteligence Applications Institute Edinburgh, May '96.

1.7.2 Books Used

Object-Oriented Specification and Design with C++
Peter Henderson, 1993, University Press Cambridge.

Turbo C++ Version 3.0 User's Guide
Borland International, 1992.

1.7.3 WWW Resources Used

wxWindows Homepage.
http://www.aiai.ed.ac.uk/~jacs/wxwin.html Nov. '96

Introduction to the Booch Method.
http://www.itr.ch/courses/case/BoochReference/ Nov. '96

Relational Rose / C++.
http://www.relational.com/pst/products/rosecpp.html Nov. '96

1.8. Glossary

C.A.S.E	Computer Aided Software Engineering.
Class	An entity similar to C's struct and union which combines functions and data.
Encapsulation	This is the welding of code and data into a single class type object[1].
G.U.I	Graphical User Interface.
Hungarian notation	A common style of varaible and object naming allowing ease of understanding.
Inheritance & Multiple-inheritance	Classes may inherit properties of other many classes. A parent class is a class from which the current class inherits properties and a child class is a class which inherits properties of the current class.
Member access control	Members of classes may be private, public, or protected. A private member may only be accessed by other members in the same class. A protected member may be accessed by any member of its or its childrens classes. A puplic member may be accessed by anything.
Motif	A set tools (widgets) used for generating Graphical User Interfaces.
Napier, Small-talk, Eifell	These are other Object-Oriented languages.
Object-orientated	A programing paradigm which C++ is a member. The paradigm attempts to mimick the way we see the real world.
R.C.S	Revision Control System.

[1] Definition taken from Turbo C++ Version. 3 User's Guide pg. 127

Resworb	The name of the application. It is the reverse of browser and is named such because of the way it allows the users to do the oposite of most browsers by allowing them to browse and generate classes before compilation.
TCL / TK	A multiple platform scriting language used for createing Graphical User Interfaces (GUIs).
Virtual functions	A function which is declared in a base classes but is actually defined in a derived class.
WxWindows	A set of C++ modules to generate user intefaces with a number of widget sets including Motif.

Part 2

Changes to Project Plan

Context Survey Conclusion (Section 2.1.6)

It is not possible to manipulate the layout of the classes in Semantic Think C. This is because there is only one mouse button available which is used to display the available member functions. This is often frustrating as it is often useful for the user to position the classes in different positions.

Editres allows the user to see the resource type while the mouse button is depressed over the name of the resource. This is a novel idea and could be incorporated into a class browser where the entire function header is displayed when a mouse button is depressed over a member function.

Changes to Project Specifications

(Section 3.1.1) The Menus

Creation

When the system starts a new project will be opened automatically. Later the user may wish to discard the current project and start afresh this is achieved by the user selecting the *new* menu option. The user will then be prompted for confirmation of discarding current edits after which a blank Inheritance pad will be created. (N.B. the default base class has now been removed.)

Loading a Project

Loading a project will be as before except now the user selects a project file rather than a directory. This allows multiple projects to co-exist in a single directory, although care will have to be taken as to not inadvertently replace files in non-open projects.

Saving a Project

Because of the changes in the creation of projects the previous s*ave* menu option has been discarded and the *save as* menu option has been renamed to *save*. This is because when a new project is created the user is no longer prompted for a project name. When a project is saved two files will be saved for each class a C++ header file and a C++ code file. These files will contain structured comments which enables the files to be loaded. Also the single binary file is now a structured ASCII file. This allows the user to edit the file if wished.

Importing

A new menu option will be available to import classes from other projects. When the user selects the *import* menu option the user will be prompted to select the header file of the class to import. After which a class will be created for the given file and displayed at the top left hand corner of the screen.

(Section 3.1.2) Editing a Project

There have been two minor changes to the specification in editing a project. The first is the user will now be able to edit, rather than just add, or delete their member variables. This is preferable as it allows the user to add comments to the variables. The second change is that instead of specifying the inheritance when creating a new class by using the *Create child class* option on the context sensitive menu, a new class may be created which does not contain any inheritance and then the inheritance may be specified latter by adding parent classes (sub-classes). This is preferable for two reasons. Firstly a new base case may be created rather than the previous restriction that all classes must inherit from a predefined base class. Secondly the user may specify the inheritance at a later stage rather than at the time of creation allowing the user to create the top level class and then define the sub-classes it inherits rather than visa-versa.

(Section 3.1.3) Other Operations

An auto save option would be very useful and would allow the user to back track to a previously saved version if they make an unrecoverable mistake. This will be implemented, if time permits, using a countdown timer.

(Section 3.3.3) Hardware

Only the left-hand and right-hand mouse buttons will be used. This is for two reasons: The first is that the application should then be directly portable to Microsoft windows and use the standard Microsoft user interface; secondly it might become confusing for the user if all the mouse buttons were used. It may however, be possible for the user to specify the action of the middle mouse button using an X11 resource file optional utilities like this will only be implemented time permitting.

(Section 3.3.5) Error Handling and Extreme Conditions

In addition to the other types of errors previously mentioned the user may also have errors in the code the user enters in the user_member functions. Although it is possible to implement a code analyser for this code it would be by no means an easy task. A possible solution would be to compile the class each time the user changes some aspect of it and then display the results. This, however, could prove to be unacceptably slow and so will not be implemented.

Project Report Contents

Part 3

Project Summary

3.1 Aims and Achievements

The aim in producing Resworb is to provide experienced object-oriented programmers with an interactive tool which facilitates the design and implementation of C++ class hierarchies. The main idea is that programmers should be able to graphically create, manipulate and edit C++ classes. This tool proves to be useful in the first stages of implementing medium to large programming projects. To this end the project has been very successful.

In achieving the aim of the project a great deal of thought had to be put into the differing styles of object-oriented programming a prospective user may adopt. After all Resworb should prove to be useful to both very strict and rigorous object-oriented programmers as well as programmers who adopt a more traditional approach. This is best emphasised by example. In many programs there are certain variables which the programmer would like to be available to all parts of the program. This is frequently achieved by using global variables. However, a very strict object-oriented programmer may wish to bind this variable into every class by creating a base class which every class would inherit where this base class would contain a static version of the variable. Although Resworb can handle both approaches it is unfortunately by far easier to adopt the stricter approach while using Resworb as it is then possible to visualise it through the class hierarchy. For programmers who prefer to use global variables Resworb may be used to design the class hierarchy but afterwards the programmer must edit the code adding non-member functions and global variables. A global section could have been added to Resworb, accessible through the menu bar, which may have alleviated this problem. But this has not been implemented as Resworb is primarily a class browser before a program developing environment.

3.2 Ideas of Design

3.2.1 Modularisation

In many programs the code for the graphical user interface is in a separate module to the data storage code, as it is in this project. However, most programmers consider that each module should be self contained in a distinct number of files. This is by no means necessary, and within Resworb it is not always possible. This is because the graphical toolkit, wxWindows, has been used which provides a comprehensive set of classes to implement the graphical user interface. These classes are inherited by the classes in Resworb, which store data and its access functions relevant to Resworb while at the same time containing methods calling of wxWindows graphical methods to provide the graphical capabilities required. To this end it is apparent that code from the graphical user interface module and code from the data access module may often be found in the same files.

3.2.2 Design Decisions

A crucial aspect which was addressed at the design stage was the saving of projects. The original specification stated there would be two files for each class (a code file and a header file) and a binary file containing application specific information. This seemed too restrictive as a plain text file is easy for debugging by the implementer and easier to hack by a user. The text file now only specifies which classes in the current directory are part of the project, where on the screen each class should be displayed and what classes does each class inherit from.

Also addressed at the design stage was the problem of how to find the balance between using structured comments and code analysing. Too many structured comments would make it difficult for a user to hack, conversely too few comments would make code analysis necessary, which is extremely complicated. A balance was found where the definition of member variables were stored between two structured comments, and a structured flag between each member function. This meant that the only code analysis necessary was to extract the name of a function out of its declaration.

3.2.3 User Extensions

The browser has been implemented using motif and so it may be possible for the user to change the user interface and add extra functionality by using motif resource files. Unfortunately this option was not implemented due to time constraints.

3.3 The user interface

3.3.1 Project Space

On start-up a new project space is created. This is a large sub-window in which the user may create and view their classes. The original intention was to create a project space of variable size where the size would be dynamically adjusted as more space for the users project was required. Unfortunately dynamic resizing is not possible under wxWindows and the maximum size of a panel is the size of the screen. This causes problems when user_classes are larger than the size of the panel. If this happens, the user should try to move the user_class upwards. By doing this the class will be snapped to a viewable position within the panel and then the show / hide member functions menu option may be selected.

3.3.2 Context Menus

The user interface was initially designed for experienced computer users who may require complex tools. However, the browser has been implemented giving the users the power required while leaving the user interface relatively clutter free. The user interface relies heavily on context sensitive menus. This is primarily because it is easy to use as there can be no confusion over whether to select the object first and then the action, or whether to specify the action and then to specify the object on which the action is to be performed, as with other menuing systems.

Context sensitive menus are also ideal in object oriented programs because the target for the action is always the current object and the action to be performed should be a member function. However, a programming bug which did arise early in the implementation process was a confusion as to which context sensitive menu should be displayed. This was due to an item with a context sensitive menu being placed on top of another item which also contained a context sensitive menu. This bug has been fixed although it is still possible to get the wrong menu when pressing the right-hand mouse button over a newly created class. A feasible solution to this bug would be to stop using context sensitive menus to create user_classes. The creation of new classes could then be achieved by using double mouse clicks.

3.3.3 Short Cuts

There are short cuts available for all items on the file menu. It would have been good to avoid having to use the context sensitive menu to do common operations such as creating a class in the project window or editing a member function while a double click on the mouse should suffice. This does not seem possible as motif captures double click events as two single clicks.

3.3.4 Inheritance Arrows

To show inheritance within a class hierarchy it was concluded that arrows should be used between the parent class and the child class. This seems confusing to many people as to which way the arrow should point. One suggested solution was to place a visible reminder within the program, however, this was not implemented because a new sub-window would have to be created to display this and in doing so the G.U.I. would become unbalanced and so less aesthetically pleasing. To rectify this would take a long time as the entire graphical user interface would have to be rewritten, due to time constraints this has not been implemented.

Another problem which arose was the positioning of the arrowheads. It was decided that positioning the arrowheads in the middle of the line would be far easier than calculating the intersection between the line and the class window. The drawing of the arrowheads was also by no means trivial as complicated mathematical functions should be avoided in frequently called functions. Although the arrowheads could have been simply implemented using such functions a more complicated option was used involving Pythagoras' theorem.

Another problem which had to be addressed was how do programmers perceive inheritance and in which direction do they program. i.e. do they make the super-class first and then the sub-classes, which the super-class inherits, or visa-versa. When specifying inheritance the browser expects parents (sub-classes) to be added to children (super-classes). It would be preferable to have both options available but this is not possible due to the way the inheritance tree is traversed.

3.3.5 Editing Member Variables

When a user specifies that they want to edit the member variables all the member variables are loaded into an edit window. The user may then use this window to add, remove or change access permissions of member variables. When the user selects "OK" the edit window is closed and the member variables for the class are updated from the new contents of the edit window. The contents of the window are not checked for errors as this may be extremely complicated. Member variables are by default private unless otherwise stated.

3.3.6 Member Functions

Each time a member function is committed after editing the function name is extracted so that it may be displayed in the class window. The extraction of the function name from the function body can be very complicated and one restriction is that pointers to functions may not be returned as the procedure which extracts the function names will not accept spaces or braces in return types. However, it is possible to get around this problem by using typedefs to define the return type. This, however, is not too great a restriction as returning functions are rarely used and even sometimes discouraged.

3.3.7 Separating

When a class is separated it is disconnected from all of its parents, this allows its (and its children's) inheritance to be redefined. This is often useful as it permits new classes to be inserted into an existing inheritance tree.

3.3.8 Response Times

A common complaint is that screen updates are slow especially when scrolling. This is due to wxWindows, the object-oriented extension of motif. wxWindows was chosen as the toolkit to use while implementing the user interface primarily because it is extremely portable between operating systems and widget sets. This enables Resworb to theoretically compile under most UNIX operating systems, including Linux, using motif or XView as the widget sets, making its portability an advantage.

Resworb would also compile on PC's using Microsoft Windows 95. In retrospect the use of wxWindows was a poor choice. Although wxWindows is actually very good and achieves its aim as a multiple operating systems G.U.I. development tool, it is however very slow, to the extent that Resworb could run faster if the G.U.I. was implemented from raw motif. Also, wxWindows contains some strange peculiarities which did not always make sense. For example a panel is not scrollable and a canvas cannot contain any widgets. Because a panel is not scrollable when the scroll bars are used each user_class is moved individually and with large projects this may become very jerky.

3.4 Special Algorithms and Data structures.

3.4.1 Using Multiple Inheritance

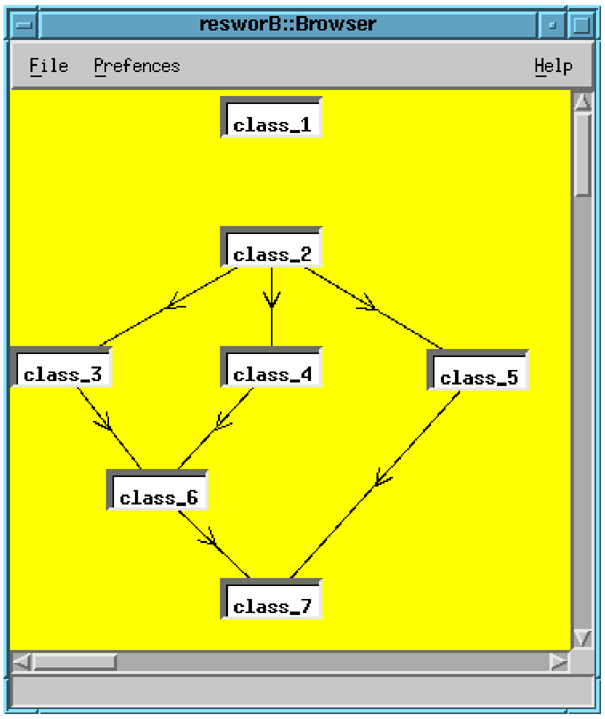
Originally when data in an object was to be edited the entire object would have to be passed to the edit_window class. Only in this way could the data be safely edited using the access functions. This was irritating, as for each type of object which contained data to be manipulated it was necessary to have a separate implementation of the edit_window class. This problem was eventually overcome using multiple inheritance with the class CData which contains virtual functions which must be over ridden by the inheriting class. This enabled only one instance of the edit_window to be implemented. Also by using virtual functions in this way when the changes in the edit_window are committed different types of analysis may be performed on the data being committed depending on the class for which the data is for, for example extracting the function name.

3.4.2 User Inheritance

The original plan was to give the user as much freedom as possible by doing little or no sanity checking. It later became apparent that the browser could become very confused if there were no restrictions on inheritance when it came to inheritance tree traversal. The first step was to disallow cyclic inheritance. This was achieved by getting the potential child to ask all of its off-spring if the potential parent is an offspring of the child. Here is the section of code which was devised to do just this :-

```
Bool CClassWindow::IsChild( CClassWindow* pSearch )
{
        CRelatedList* pTchildren;
        CClassWindow* pChild;
        Bool ret = FALSE;
        pTchildren = m_pChildren;
        while ((pTchildren!=NULL)&& (ret==FALSE))
        {
                pChild = pTchildren->GetClass();
                if (pChild == pSearch ) return TRUE;
                ret = pChild->IsChild( pSearch );
                pTchildren = pTchildren->GetNext();
        }
        return ret;
}
```

In essence this algorithm steps along its list of children asking the question "Are you the potential parent?" If the answer in negative then the child asks all of its children the same question recursively until there are no more children or a positive answer is given. (For a fuller explanation see section 3.4.3.) The complexity of this is approximately O(n) where n is the number of user_classes as each class is only addressed once unless multiple inheritance is used.



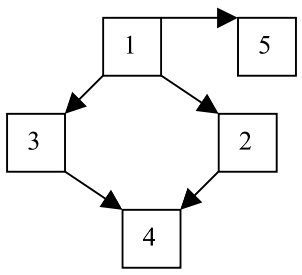
If class 2 required to inherit class 1 then class 2 will ask all of its descendants if they are class 1. In doing this class 6 will be asked twice and class 7 will be asked 3 times.

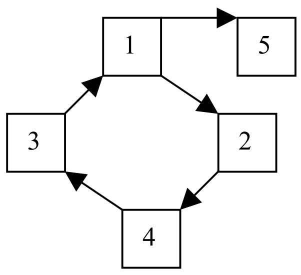
It was also at this time that a contemporary suggested banning multiple inheritance as they claimed it was rarely used properly and causes many problems. However, multiple inheritance has been used in creating this browser and so it should be available to its users.

3.4.3 Storing the users hierarchy

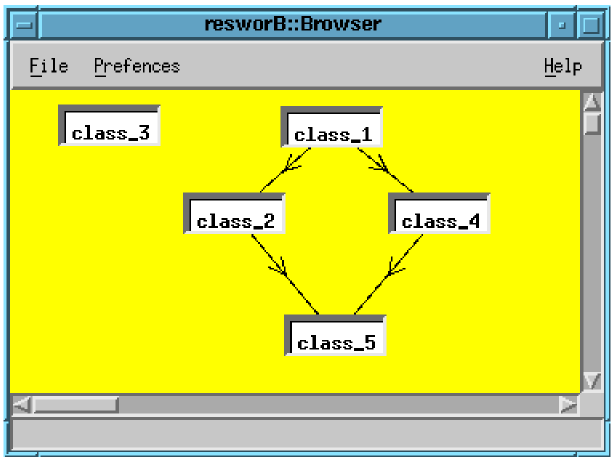
The user_class hierarchy is very similar to a non-cyclic directed graph. So for example

This is valid And this is not because it contains a cycle

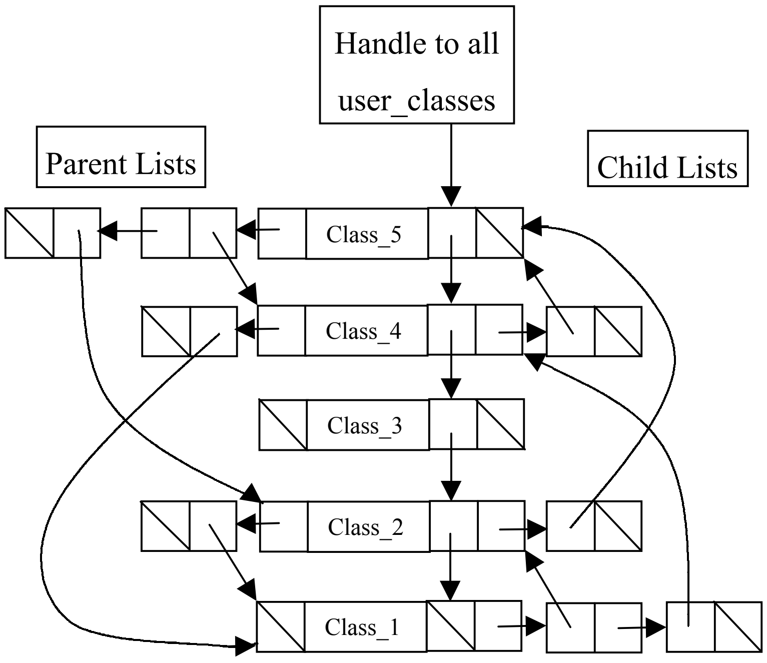




The user_classes are similar to the nodes on the graph except user_classes also store data about their member variables and member functions. For each user_class to keep track of its parents and its children, lists of pointers are used. Each user_class maintains two lists and one pointer. The first list contains pointers to all of the class' children; the second list contains pointers to all of the class' parents and the pointer points to the previously created user_class so that all classes may be easy accessed without performing a complicated traversal. For example :-



this class hierarchy would be stored in memory like this.



As can be seen the single pointer which points to the previously created class makes up a list of all classes where each class is only on the list once. This is ideal for traversing all user_classes. The list pointing to class' parents is necessary so that the inheritance links may be drawn while the list pointing to the classes children is necessary to test if a potential parent is already a child (see section 3.4.2).

3.5 Conclusion

Overall the project was successful it achieved all of its primary aims and implemented some of its secondary ones. The implementation has been well structured making future enhancements trivial given time. With hard work Resworb may even be extended to become a complete program development tool.

Resworb provides a clean and simple user interface making it very pleasant and easy to use. Unfortunately using wxWindows makes the graphics a little slow. However, Resworb is able to browse its own class hierarchy which proves its effectiveness and I certainly will continue developing future versions of Resworb and use Resworb to develop other C++ applications.

Part 4

Testing Summary

The testing had to be carried out in many stages, due to the way Resworb fits together as an object oriented program, making it impossible to test each module independently. For example the program_class which stores user_classes inherits a graphical class and so the storage of user_classes could not be tested without using the G.U.I module. Therefore a very rigorous and methodical approach was necessary to extensively test each module. This was achieved by retesting the module currently being implemented after each significant function was completed. The tests were broken into five distinct stages: The first stage was using one piece of valid data, the second using many pieces of valid data and the third using no data. The fourth stage intentionally used invalid data in an attempt to break the procedure and the fifth stage involved contemporaries using the program. Testing each significant function in each module this way does not prove that the function is sound and correct although it does give a very good indication. After all the modules were complete the entire project was extensively tested using a similar method where both valid and invalid data were used.

For example when the drag & drop of classes was being tested during the implementation of the user interface module. The first stage of the test was to see if a class could be moved without touching the edge of the window. The second was to ensure that when an edge was encountered the class stopped moving. The third stage was to assert that the class would stop moving when two edges were encountered simultaneously (i.e. when a corner was encountered). The forth stage used a class which was larger than the window so that one part of the class window must be off the viewable inheritance pad. The final stage of the test was to get other people to test drag and drop.

After all the modules had been implemented, the entire project needed to be tested exhaustively this was achieved by using Resworb to create, load and save projects. The most complete of these tests was to load the source code for Resworb into Resworb allowing the projects class hierarchy to be viewed.

Part 5
Status Report

The current state of Resworb is that all of the specified functions work, although possible not in the same way as specified in the plans functional requirements (section 1.3.1). However, none of the possible modifications stated in the project plan have been implemented due to lack of time. However importing functions have been added.

Creating User Classes

The user may create or import classes into the current project.

Editing User Classes

Member Functions :- Member functions may be added and deleted from user_classes. They may not be copied between classes. A restriction with member functions is that the function declaration must be the first line of the function in the edit window and the member function may not return pointers to functions unless achieved using either *#define*s or *typedef*s.

Editing member variables :- Member functions may be edited within the browser. There are two restrictions; one the first line in the edit function window must be the function declaration. This is so that the function name may easily be extracted. The second restriction is that pointers to functions may not be the returned from functions this also is due to extracting the name of the function from its definition.

Changing Inheritance :- Inheritance may added and removed from classes.

Move :- Classes may be moved anywhere within the graphics pad.

Deleting of Classes :- User_class may deleted in two ways. The entire tree below the current may be deleted or just the current class may be deleted with its children inheriting properties of the current classes parents.

Save :- As well as entire projects being saved it is possible for just the current class to be saved. This an ideal way of making back ups as it is not possible undo actions.

User Projects

New :- The current project may be discarded, giving the user a clear inheritance pad.

Save :- The current project may be saved to a single directory in standard C++ format.

Open :- A previously saved project and a project the user has created manually may be loaded into a graphics pad with all previous user_classes being deleted.

Importing :- A previously saved project may be imported into the current project.

The Graphical User Interface

The user interface is easy to use and clutter free. However, there is no option to snap the user classes to a grid. The program window may be resized and the inheritance pad is scrollable. However the inheritance pad is only of finite size.

Part 6 Technical Documentation

Resworb

User Manual

Contents

Before You Start

Requirements

This program is designed to run on the DEC Alpha Workstations in the Philip Lee laboratory at the university of St Andrews using a Colour monitor, a two button mouse and keyboard. It will also run on Linux machines and under Windows95.

Using This Manual

It is also assumed that the reader is familiar with C++ programming and the UNIX operating systems and using graphical user interfaces.

It is not necessary to read this manual from beginning to end, however to gain a full understanding of the application a study of this manual should be made.

This manual has been designed as an extended example starting with installing Resworb to creating and finishing a project.

Typefaces used in this Manual

[ ] square brackets denote optional parameters to programs

Italics. Italics are used for smaller sections of example code.

< > Angle brackets denote key presses on the keyboard.

Compiling Resworb

Assuming the wxWindows library has been successfully compiled. Resworb may be compiled by changing to the directory in which the Resworb code is stored and typing *make* <Return>. It may be necessary to edit the Makefile according to your system. Once the Resworb binary has been built it may then be copied to a directory within the path.

Starting Resworb

Within a Unix windowing session open an command terminal

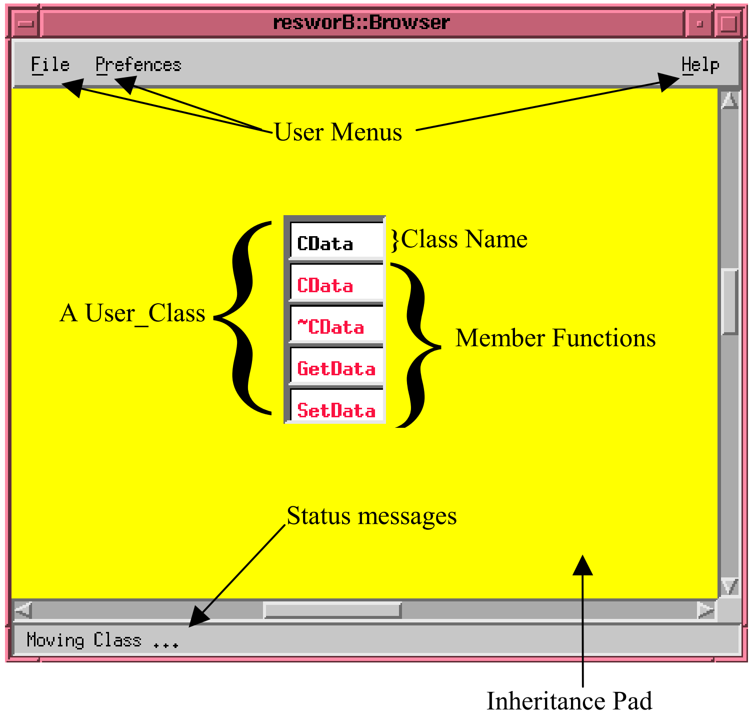
Then if Resworb binary is in the users path type *Resworb* <Return>

Else change directory to where the binary may be found and then repeat the last step.

N.B. If Resworb is in the users path then a project may be opened with by changing to the project directory and typing

Resworb [project_name].rwb

The Main Window



Once you have followed the steps in *Starting Resworb* a window similar to above will be displayed.

Inheritance Pad

The inheritance pad is the large area of the window in which the users classes are created and displayed.

Using Resworb

Creating Classes

The first thing a user would wish to do is create a new class. This is achieved by the user depressing the right mouse button over the inheritance pad. A pop-up menu will then displayed giving the user to create a new class where the mouse pointer is.

The new class will be created with a default name at the position specified.

Moving Classes

A class may be moved by holding the left mouse button down over the class and dragging to the new position and then releasing the mouse button.

Editing Classes

Once the user has created the class, properties may be assigned to the class. The first thing the user will wish to change is the class name. This is achieved moving the mouse pointer over the class name and then depressing the right mouse button to display a pop-up menu. The user may then select the *Edit class name* option. After which a message box will be displayed prompting for the user to enter the new class name. Following this the user may wish to give the class a description. This also is achieved in a similar manor.

Member Functions

Member functions may be added to by selecting the *add member function* option from the pop-up menu when the right mouse button is depressed over the class name. To delete member functions depress the right mouse button over the function to be removed and select the *delete* option. In the same way the member access control may be specified.

When editing functions it is important to remember three things :-

1) The function definition must be on the first line
2) Pointers to functions may not be returned unless by using typedefs
3) Do insert the class name in the function definition Resworb will do it for you.

Inheritance

When user wishes to inherit the properties of one class into another the user. It is achieved by the user bringing up the pop-up menu over the class name and selecting the *add parent* option. The mouse pointer will then change into a cross-hair until a valid parent class has been selected. Removing inheritance is achieved in much the same way except that all of the class inheritance is removed.

Deleting Classes

Deleting classes should be done with care as there is no undo option available. There are two forms of delete. The first deletes the current class and all child-classes (delete branch). The second just deletes the current class and makes the current class' children point to the current class' parent if one exists.

Saving

There are two forms of saving the first is the saving of the current project and the second is the saving of the current class. To save the current project the user selects the *save* option from the *file* menu or the user can press <Control + S>. In both cases a dialog box will appear prompting the user to enter the name of the project and the directory in which it is to be stored.

N.B. care must be taken here as existing file in the same directory may be over wrote if the files are the same name as a class.

To store just a single class, simple select the save option from the class names pop-up menu. The class is then saved to the current directory.

Loading

A project may be loaded in two ways. The first is by specifying the project as a command line parameter when starting Resworb and the second is by using the *load* option from the *file* menu also accessed by pressing <Control + L>. In this case the user will be prompted whether or not to discard the current inheritance pad then the user must select a project to load.

Importing

An entire project may be imported into the current project. This is useful if the user wishes to import an entire class hierarchy, achieved through the file menu. The drawback of this is that there is no intelligent placing and imported class may be displayed on top of current classes and imported classes with the same name as currently loaded classes will not be imported.

Single classes may also be imported into a project. When a single class is imported it is places at the top left of the screen and all its information about inheritance is lost from Resworb.

User Preferences

The users preferences are stored in a file in the users home directory. These preferences may be edited through the preferences menu. The first line of the preferences file contains brief instructions on how the preferences may be changed.

Importing External Projects

C++ class hierarchies which have not been created using Resworb may be viewed in Resworb with a little manipulation by the user. This is best achieved by the user editing each class so that it may be imported by Resworb and then the user specifying the inheritance of the classes once in Resworb. This may be done in a number of stages.

1) In before each function in the code files insert //#<access_permissions> for example :-

//#PUBLIC	for a member function which has public access.

2) At the end of the code file append a *//#Finish* then a few blank lines.
3) In each header file the start of the member variables should be flagged by a *//#Start* and the end of the member variables should be flagged by a *//#Finish*.

Part 7
Maintenance Document

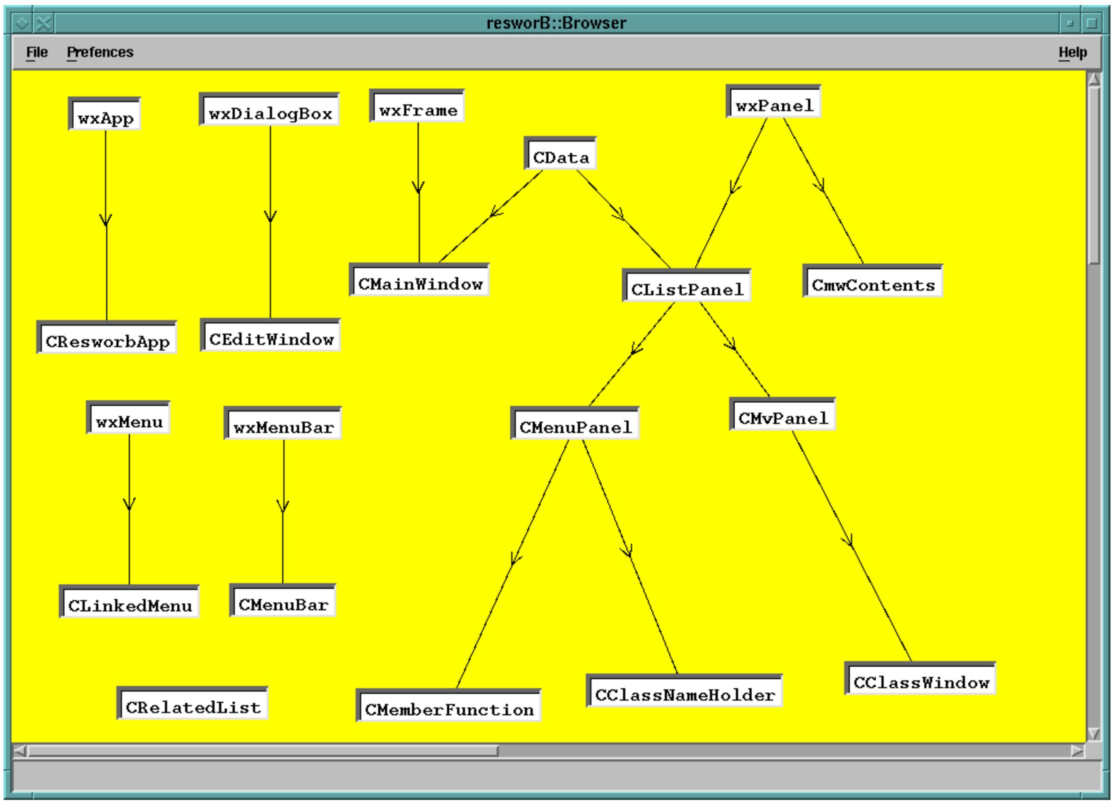
7.1 The Modularisation

Due to the way Resworb has been implemented it is difficult to perceive how modularisation has been achieved. This is because Resworb is made up of classes and each class may contain code for a number of modules. The two key modules are the graphical user interface module and the user_class module. In general the code for the graphical user interface module will be found near the start of the class code file while the code for the user_class module may be found towards the end of the class code file.

7.2 The Class Hierarchy

The source code is well commented but just to clarify things here is the Resworb class Hierarchy. The key Resworb classes are briefly summarised in this section.

N.B. All classes from the wxWindows toolkit start with "wx" while all classes belonging to Resworb start with "C".



7.2.1 Class CResworbApp

All programs which use the wxWindows toolkit must contain an object (not a pointer to an object) of a class which inherits the wxApp class. The name of the object is *ResworbApp*. The member function *OnInit()* is similar to the *main* function in other programs and instantiates the application.

The function *OnInit* creates the global variables and constants and also creates and returns the application window.

7.2.2 Class CMainWindow

The class CMainWindow creates and manages the main window to the application. It contains a pointer to the applications pulldown menus (CMenuBar) and also a pointer to the inheritance pad. The functions which are called through the pulldown menus are also in this class.

7.2.3 Class CEditWindow

This is the generic class through which all data is edited. The class constructor takes as a parameter the class CData which all classes which contain data inherit. In this way all classes which contain data may be passed to this class to be edited.

7.2.4 Class CData

This class is inherited by all classes which contain data. In this way these classes may be passed to the class CEditWindow. This class contains a virtual functions *SetData()* which is redefined as necessary in each class which inherits CData. The function *SetData()* is called by CEditWindow, hence it must be virtual.

7.2.5 Class CLinkedMenu

Originally this was not implemented but it became apparent that a method was needed for accessing the object which called the pop-up menu. The menus are not created in this class as they are created in the classes which call the menu. But the function which is called after a menu event, *PopupFunction*, is here. This function deciphers what menu event occurred using a large switch statement and then calls the relevant member function.

7.2.6 Class CRelatedList

A user_class may contain many parents and children. The parents and children of a user_class are stored in a list of pointers which is implemented using the class CRelatedList.

7.2.7 Class CClassWindow

This is the largest and most complex class within the entire application. Each object of this class maintains an entire user_class. It points to its member functions, member variables and all of its relations. The class contains code from four modules. The first part of the file contains entirely graphical code for displaying inheritance arrows, function names and class names. The next section of the file contains the entire graphs module which maintains the parent and child lists and it also checks for cyclic inheritance. The third section contains code from the classes module which maintains the users_class data. The last section of the code file contains code from the files module. This sections contains code to load and save the user_class.

7.3 Guide to Further Developments

The program may be changed in two ways. Firstly, bugs may be fixed, and secondly new features may be added. This section is a guide to how to go about making these changes.

7.3.1 Recommended Changes

Here is a list of changes which should be made to the code given time.

1) The class CClassWindow should be broken down into a number of classes. Firstly a class dealing with a directional graphs.

   Secondly a class which contains and edits the data

   Thirdly a Class which deals with the loading and saving of data.
2) Extending inheritance so that it may be either public, private or protected.
3) Loading and saving does more code analysis to determine its inheritance so that when a class is imported its hierarchy may also be imported.
4) Support for global variables and non-member functions.

7.3.2 Adding New Features

There are many features which could be added to Resworb and it would be impossible to cover all possibilities here. Before a feature is added I would recommend that the implementers familiarises themselves with the structure of Resworb, and note where they feel the new feature would fit. The next question would be "How would the user perceive this new feature?", as this is often a more realistic indication as to where the new code should fit. After a new feature is added it is very important that the entire application is retested.

7.3.3 Fixing bugs

In the unlikely event of a bug becoming apparent then the first step should be to fully document the bug describing exactly what happens and in what circumstances it occurs. The next step is to see if the bug still exists when a debug version of Resworb is ran through a debugger. If this is the case then the bug may easily be tracked down and fixed. If on the other hand the bug is no longer apparent then add comments around the code where it is suspected the bug may be and add lots of print statements which may be used to track the programs execution.

7.3.4 Rebuilding & Testing the program after Developing

The version of wxWindows which was used is 1.66b , this may be compiled by following the installation instructions provided with wxWindows. However, it was noticed that there is a bug in their code for motif pop-up menus. The fix may also be down loaded from the wxWindows web site. Once wxWindows has been built the compilation of Resworb is trivial. First edit the Makefile to point to the relevant libraries and include paths, then just type *make*.

Part 8
Software Listings

Contents

8.1 App_Def.h

```
//********************************* App_Def.H *************
// Application default settings file
// 21/12/96
//*******************************************************
//#include <stdlib.h>
#include <stdio.h>

#ifndef DEFAULTS_H
#define DEFAULTS_H

#define MAX(a,b) ((a < b)?b:a)

extern wxColour* g_pColBLACK;
extern wxColour* g_pColWHITE;

extern wxColour* g_pColBGround;  // Background colour
extern wxColour* g_pColProtected;  // Protected member function colour
extern wxColour* g_pColPrivate;  // privated "          "          "          "
extern wxColour* g_pColPublic;  // public               "          "          "

// maximum length of classname
extern int g_MaxNameLen;
// maximum length of window claculated from name lengh and font width
extern int g_WindowWidth;
extern int g_SectionHeight;
extern int g_MaxDescriptionLen;

// Global application font
extern wxFont* g_pAppFont;

// canvas popup menu
#define NEW_BASE_CLASS 1
// defines for classname pop-up menu
#define ADD_FUNCTION 10
#define ADD_VARIABLE 11
#define EDIT_DESC 12
#define CUT 13
#define DELETE_BRANCH 14
#define DELETE_CLASS 15
#define EDIT_NAME 16
#define ADD_PARENT 17
#define VIEW_MEMBERS 18
#define SAVE_CLASS 19

// file_menu
#define LOAD_PROJECT 20
#define SAVE_PROJECT 21
#define NEW_PROJECT 22
#define QUIT_RESWORB 23
#define EDIT_PREFS 24
#define ABOUT_RESWORB 25
#define SHOW_HELP 26
#define  IMPORT_CLASS 27
#define IMPORT_PROJECT 28
```

```
// defines for member function pop-up menu
#define EDIT_FUNCTION 29
#define DELETE_FUNCTION 30
#define MAKE_PUBLIC 31
#define MAKE_PROTECTED 32
#define MAKE_PRIVATE 33

#define PRIVATE 34
#define PROTECTED 35
#define PUBLIC 36


#endif
```

8.2 Assert.h

```
//******************************** Assert.H *************
// Cause asserts
// 24/12/96
//*****************************************************
#ifndef ASSERT_H
#define ASSERT_H

int AssertCrash( void );

// used to test if something is true. If not quit application
// not used in relase version.
#define assert(exp) (void)( exp ? 0 : AssertCrash())


#endif
```

8.3 Assert.cpp

```
//********************************** Assert.cpp *************
// causes application to crash after asserting
// 24/12/96
//*******************************************************

#include <wx_prec.h>
#include <wx.h>

int AssertCrash( void )
{
        wxMessageBox("Assertion Failure", "Assertion Failure");
        // its harder than it seems getting the application to die fast!
        // but this work very well.
        wxButton* die = NULL;
        die->SetLabel( "Just some jibberish to kill the app");
        return 0;
};
```

8.4 CClassNameHolder.h

```
//**************************** clname.h *****************
//
//                    23/11/96   code for managing class names bindings
//                                         and display
//                    Martin Bramley
//******************************************************

#ifndef UNIX
#include<wx_prec.h>
#endif

#include<wx.h>

#include "app_def.h"

#ifndef CLASSNAME_H
#define CLASSNAME_H

#include "CMenuPanel.h"

class CClassNameHolder: public CMenuPanel
{
//#Start
private:
        wxPanel* m_pParent;
//#Finish
public:
        CClassNameHolder( wxPanel*,int,int,int,int);
        ~CClassNameHolder( void );
//      wxPanel* GetParent( void );
        void GetDims(float&, float&);
        void OnPaint( void );
        void EditData( void );
        void SetData( const char* );
};

#endif
```

8.5 CClassNameHolder.cpp

```
//***************************** clname.cpp ***************
//
//                  23/11/96   code for managing class names bindings
//                                  and display
//                  Martin Bramley
//*****************************************************
#include <stdlib.h>
#include <stdio.h>

#include <string.h>
#include <wx_prec.h>
#include <wx.h>

#include "CClassWindow.h"
#include "CClassNameHolder.h"
#include "CEditWindow.h"

const char* pConstDefaultDesc= "Default Class Description \
                                        \n Maximum length is 512 charaxters\n";

void MakeClassMenu( CLinkedMenu* pMenu )
{
        pMenu->Append( VIEW_MEMBERS, "Show/Hide Member Functions");
        pMenu->Append( ADD_FUNCTION, "Add Member Function");
        pMenu->Append( ADD_VARIABLE, "Edit Member Variables");
        pMenu->AppendSeparator();
        pMenu->Append( EDIT_NAME, "Edit Class Name");
        pMenu->Append( EDIT_DESC, "Edit Class Description");
        pMenu->Append( ADD_PARENT, "Add Parent");
        pMenu->Append( CUT, "Separate");
        pMenu->AppendSeparator();
        pMenu->Append( SAVE_CLASS, "Save Class");
        pMenu->Append( DELETE_BRANCH, "Delete Branch");
        pMenu->Append( DELETE_CLASS, "Delete Class");
}

//#PUBLIC
CClassNameHolder::CClassNameHolder( wxPanel *parent, int x=-1, int y=-1,
         int w=-1, int h=-1) :  CMenuPanel( parent, x,y,w,h)
{
        m_pParent = parent;
        MakeClassMenu( m_pMenu );

        SetData( pConstDefaultDesc);
}

//#PUBLIC
CClassNameHolder::~CClassNameHolder( void )
{ }

/*wxPanel* CClassNameHolder::GetParent( void )
{
                return m_pParent;
}*/
```

```
//#PUBLIC
void CClassNameHolder::GetDims( float& w, float& h)
{
	char* clName = ((CClassWindow*) GetParent())->GetClassName();
	wxDC* pDC = GetPanelDC();
	pDC->GetTextExtent( clName, &w, &h);
}

//#PUBLIC
void CClassNameHolder::OnPaint( void )
{
	wxDC* pDC = GetPanelDC();
	pDC->Clear();
	int w=0,h=0;
	GetSize( &w, &h);
	pDC->DrawRectangle((float) 0, (float) 0,(float) w, (float) h);
	pDC->DrawText( ((CClassWindow*) m_pParent)->GetClassName(), (float) 5,
		 (float) ((g_SectionHeight /2) - (g_SectionHeight /4)));
}

//#PUBLIC
void CClassNameHolder::EditData( void )
{
	CEditWindow* pEdit = new CEditWindow( NULL, this,
		"Edit Class Description",100,100,400,400);
	pEdit->Show(TRUE);
}

//#PUBLIC
void CClassNameHolder::SetData( const char* pDesc )
{
	if (m_pData != NULL)
		free( m_pData);
	m_pData = strdup( pDesc );
}

//#Finish

// last line
```

8.6 CClassWindow.h

```
//**************************** classwin.h ******************
//
//			23/11/96   code for managing class names bindings
//						and display
//			Martin Bramley
//******************************************************

#ifndef UNIX
#include <wx_prec.h>
#endif

#include <wx.h>
#include <stdlib.h>


#ifndef CLASSWINDOW_H
#define CLASSWINDOW_H

#include "app_def.h"
#include "CMvPanel.h"
#include "CClassNameHolder.h"
#include "CMemberFunction.h"
#include "CRelatedList.h"

class CClassWindow: public CMvPanel
{
//#Start
private:
		char* m_pClassName;
		int m_currentHeight;
		CMemberFunction* m_pMemberFuns;
		CRelatedList* m_pAdults;
		CRelatedList* m_pChildren;
public:
		CClassNameHolder* m_pCNHolder;
		Bool m_SeeMembers;
//#Finish
public:
		CClassWindow( wxPanel*);
		~CClassWindow( void );
		void Display(int,int,float,float);
		void Display(int,int);
		void Display(void);
		void OnPaint( void );
		void UpdateChildLinks( int,int,float,float);
		void AddFunction( void );
		char* GetClassName( void );
		void ChangeClassName( void );
		//void CreateChild( void );
		void SetClassName( char* );
		void NewClass( void );
		void AddParent( CClassWindow*);
		void AddChild( CClassWindow*);
		void RemoveParent( CClassWindow* );
		void RemoveChild( CClassWindow* );
		Bool DeletingClass( void );
		Bool DeletingTree( void );
		void DeleteFunction( CMemberFunction* );
		void RecursiveDelete( CClassWindow* );
		Bool HasParent( void );
		void ChangeInheritance(void);
		void Cut(void);
		void OnEvent( wxMouseEvent& event );
		Bool IsChild( CClassWindow* );
		Bool SaveMe( void );
		Bool LoadMe( void );
		int GetParentList( char* );
		void AddInheritance( void );
		int GetChildList( char* );
		void SetData( const char* );
		void EditMembers( void );
		void SetViewable( void );
//		Bool IsViewable( void );
};
#endif
```

8.7 CClassWindow.cpp

```
//***************************** classwin.cpp ***************
//
//                  23/11/96   code for managing class names bindings
//                                      and display
//                  Martin Bramley
//******************************************************

#include <string.h>
#include <wx_prec.h>
#include <wx.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#include "assert.h"
#include "app_def.h"
#include "CClassWindow.h"
#include "CMemberFunction.h"
#include "CmwContents.h"
#include "CMainWindow.h"
#include "CRelatedList.h"
#include "CEditWindow.h"

extern CMvPanel* g_pMover;
extern float dx;
extern float dy;
// pointer to class which is doing the inheriting.
// user selects add parant.
// inheritor set to currentclass
// user select new parent
// class detects inheritor is not null so it is becoming a parent…..Q.E.D
CClassWindow* g_pInheritor=NULL;

// Function to read a line of text from a file.
void GetLine(FILE* inFile, char* pString)
{
        int i=0;
        int c;

        c=fgetc( inFile );
        while( c!='\n')
        {
                pString[i] = char( c );
                i++;
                c=fgetc( inFile );
        }
        if feof( inFile )
                exit( -1 );

        pString[i] = '\0';

}


Bool FlagFound( char* pLine )
```

```
{
        if ((0!=strcmp( pLine, "//#Start")) && (0!=strcmp( pLine, "//#PRIVATE"))
                 && (0!=strcmp( pLine, "//#PROTECTED")) && (0!=strcmp( pLine,
                  "//#PUBLIC")) && (0!=strcmp( pLine, "//#Finish")))
                return FALSE;
        return TRUE;
}

//#PUBLIC
CClassWindow::CClassWindow( wxPanel *parent) : CMvPanel(parent)
{
        m_pClassName = (char*) malloc( g_MaxNameLen);

        sprintf( m_pClassName, "NoName%d", GetID());

        m_currentHeight = g_SectionHeight;
#ifndef wx_msw
        SetSize( 1, 1);
#else
        SetSize(0,0,1,1);
#endif

        m_pCNHolder = new CClassNameHolder( (wxPanel*) this,0,0,(int)
                g_WindowWidth, g_SectionHeight);
        m_pMemberFuns = NULL;
        SetPen( new wxPen( *g_pColBLACK, 3, wxSOLID));

        m_SeeMembers = FALSE;
        m_pAdults = NULL;
        m_pChildren = NULL;
        SetData("//Just enter the member variables in the same way \
                \n//As in a class declaration \
                \n//do not enter any functions declaration \
                \n//e.g.\t public:\n//\t\tchar* pAString;\n");
}

//#PUBLIC
CClassWindow::~CClassWindow(void)
{
        //while (((CClassWindow*) GetNext())!=NULL)
        //      delete ((CClassWindow*) GetNext());
        delete m_pCNHolder;
        delete m_pAdults;
        delete m_pChildren;
        delete m_pMemberFuns;
        free( (void*) m_pClassName);
        ((CmwContents*)m_pParent)->RemoveClass( this );
}

//#PUBLIC
void CClassWindow::SetViewable()
{
        if (m_SeeMembers)
                m_SeeMembers = FALSE;
        else
                m_SeeMembers = TRUE;

        CMemberFunction* pFun;
        pFun = m_pMemberFuns;
```

```
	while (pFun != NULL)
	{
		pFun->Show(m_SeeMembers);
		pFun = pFun->GetNext();
	}
	Display();
}


//#PUBLIC
Bool CClassWindow::DeletingClass()
{
	CRelatedList* pTadults  = NULL;
	CRelatedList* pTchildren = NULL;
	CClassWindow* pParent = NULL;
	CClassWindow* pChild = NULL;
	// add classes children to classes parents!
	pTadults = m_pAdults;
	while (pTadults != NULL)
	{
		pParent = pTadults->GetClass();
		pTchildren = m_pChildren;
		while (pTchildren != NULL)
		{
			pChild = pTchildren->GetClass();
			pParent->AddChild( pChild );
			pTchildren = pTchildren->GetNext();
		}
		pTadults = pTadults->GetNext();
	}
	pTchildren = m_pChildren;
	while (pTchildren != NULL)
	{
		pChild = pTchildren->GetClass();
		pTadults = m_pAdults;
		while (pTadults != NULL)
		{
			pParent = pTadults->GetClass();
			pChild->AddParent( pParent );
			pTadults = pTadults->GetNext();
		}
		pTchildren = pTchildren->GetNext();
	}
	pTadults = m_pAdults;
	while (pTadults != NULL)
	{
		pParent = pTadults->GetClass();
		pParent->RemoveChild( this );
		pTadults = pTadults->GetNext();
	}
	pTchildren = m_pChildren;
	while (pTchildren != NULL)
	{
		pChild = pTchildren->GetClass();
		pChild->RemoveParent( this );
		pTchildren = pTchildren->GetNext();
	}
	return TRUE;
}
```

```
//#PUBLIC
Bool CClassWindow::HasParent( void )
{
        if (m_pAdults == NULL)
                return FALSE;
        return TRUE;
}

//#PUBLIC
void CClassWindow::DeleteFunction( CMemberFunction* pFun)
{
// because of updating pointers withinthe list of functions a function may not delete its self. This could
be rectified by using a doubly linked list.
        CMemberFunction* pTemp;
        pTemp = m_pMemberFuns;
        if (m_pMemberFuns == pFun) {
                m_pMemberFuns = m_pMemberFuns->GetNext();
                pFun->SetNext( NULL );
                m_currentHeight = m_currentHeight - g_SectionHeight;
                delete pFun;
                Display();
                return;
        } else
                while (pTemp->GetNext() != NULL) {
                        if (pTemp->GetNext() == pFun){
                                pTemp->SetNext( (pTemp->GetNext())->GetNext());
                                pFun->SetNext( NULL );
                                delete pFun;
                                m_currentHeight = m_currentHeight -
                                        g_SectionHeight;
                                Display();
                                return;
                        } else {
                                pTemp = pTemp->GetNext();
                        }
                }
}

//#PUBLIC
void CClassWindow::RecursiveDelete( CClassWindow* pDeletingParent )
{
        CRelatedList* pTadults  = NULL;
        CClassWindow* pParent = NULL;
        CClassWindow* pChild = NULL;

// this is really wierd! but it works call it a futuristic side-effect
        while (m_pChildren != NULL) {
                pChild = m_pChildren->GetClass();
                pChild->RecursiveDelete( this );

                if (pChild->HasParent()==FALSE)
                {
                        delete pChild;
                }
        }

        pTadults = m_pAdults;
        while (pTadults != NULL)
```

```
		{
			pParent = pTadults->GetClass();
			if ((pParent == pDeletingParent)|| (pDeletingParent == NULL))
			{
				pParent->RemoveChild( this );
				this->RemoveParent( pParent );
			}
			pTadults = pTadults->GetNext();
		}
}


//#PUBLIC
Bool CClassWindow::DeletingTree( void )
{
		RecursiveDelete(NULL);
		return TRUE;
}

//#PUBLIC
void CClassWindow::Display(void)
{
		int x,y;
		this->GetPosition( &x,&y);
		Display(x,y);
}

//#PUBLIC
void CClassWindow::Display(int x, int y)
{
		float MaxHeight, CurrentHeight;
		float MaxWidth, CurrentWidth;
		CMemberFunction* pTemp1 = m_pMemberFuns;

		m_pCNHolder->GetDims( CurrentWidth, CurrentHeight);
		MaxWidth = CurrentWidth;
		MaxHeight = 2*CurrentHeight;
if (m_SeeMembers)
		while (pTemp1 != NULL) {
			// some new code
			pTemp1->Move(-1,int(MaxHeight));
			// end of new
			pTemp1->GetDims( CurrentWidth, CurrentHeight);
			if (MaxWidth < CurrentWidth)
				MaxWidth = CurrentWidth;
			MaxHeight = MaxHeight + (2*CurrentHeight);

			pTemp1= pTemp1->GetNext();
		}

		Display(x,y, MaxWidth+15, MaxHeight);
}

//#PUBLIC
void CClassWindow::Display( int x, int y, float w, float h)
{
		((CmwContents*) m_pParent)->Clear();
#ifndef wx_msw
		SetSize(int (w), int(h));
```

```
#else
        SetSize( x,y, int(w), int(h));
#endif
        Move(x,y);
//      OnPaint();
        ((CmwContents*) m_pParent)->OnPaint();
}

//#PUBLIC
void CClassWindow::UpdateChildLinks(int x1, int y1, float w1, float h1)
{
        int x2,y2,w2,h2;
        float midx1, midx2, midy1, midy2;
        midx1 = x1 + (w1/2);
        midy1 = y1 + (h1/2);

        //variable for arrow heads
        float dx, dy, dz, midx, midy, p_dx, p_dy, n_dx, n_dy;
        float ex1, ey1, ex2, ey2;

        CRelatedList* pList;
        CClassWindow* pChild;
        pList = m_pChildren;
        while (pList != NULL)
        {
                pChild = pList->GetClass();
                pChild->GetPosition( &x2, &y2);
                pChild->GetSize( &w2, &h2);
                midx2 = x2 + float((w2/2)); midy2 = y2 + float((h2/2));

                ((CmwContents*) m_pParent)->SetSolidBlackPen();

                // parent -> child
                ((CmwContents*) m_pParent)->DrawLine( midx1, midy1,
                        midx2, midy2);

                // now for arrows
                midx = (midx1 + midx2)/2;
                midy = (midy1 + midy2)/2;
                dx = ( midx1- midx2);
                dy = ( midy1- midy2);
                dz = sqrt( dx*dx+dy*dy);
                p_dx = -10*(dx/dz);
                p_dy = -10*(dy/dz);
                n_dx = 5*(dy/dz);
                n_dy = 5*(-dx/dz);
                ex1 = midx - p_dx + n_dx; ey1 = midy - p_dy + n_dy;
                ex2 = midx - p_dx -n_dx; ey2 = midy - p_dy -n_dy;

                ((CmwContents*) m_pParent)->DrawLine( midx, midy, ex1, ey1);
                ((CmwContents*) m_pParent)->DrawLine( midx, midy, ex2, ey2);
                // end arrow code
                pList = pList->GetNext();
        }
        ((CmwContents*) m_pParent)->SetBluePen();
#ifdef wx_msw
        this->OnPaint();
#endif
}
```

```
//#PUBLIC
void CClassWindow::ChangeInheritance(void)
{
        ((CmwContents*) m_pParent)->SetCursor( new wxCursor( wxCURSOR_CROSS));
        ((CMainWindow*)((CmwContents*)(this->m_pParent))->GetParent())->
                SetStatusText("Click on Class to become parent ....");
        g_pInheritor = this;
}

//#PUBLIC
void CClassWindow::Cut (void )
{
        CClassWindow* pParent;

        while (m_pAdults != NULL)
        {
                pParent = m_pAdults->GetClass();
                pParent->RemoveChild( this );
                this->RemoveParent( pParent );
                //m_pAdults = m_pAdults->GetNext();
        }
        m_pParent->Clear();
        ((CmwContents*)m_pParent)->OnPaint();
}

//#PUBLIC
void CClassWindow::AddFunction( void )
{
        CMemberFunction* pTemp;
        pTemp = new CMemberFunction( (wxPanel*) this, -1, -1,(int)
                 g_WindowWidth, g_SectionHeight);
        pTemp->Move(0,m_currentHeight);
        m_currentHeight = m_currentHeight + g_SectionHeight;
        pTemp->SetNext( m_pMemberFuns );
        m_pMemberFuns = pTemp;
        Display();
}


//#PUBLIC
char* CClassWindow::GetClassName( void )
{
        return m_pClassName;
}

//#PUBLIC
void CClassWindow::SetClassName( char* pName )
{
        sprintf( m_pClassName, "%s", pName);
}

//#PUBLIC
void CClassWindow::ChangeClassName( void )
{
        char* pText;
        char* msg = (char*) malloc( 100 );
        sprintf( msg, "Class name must be less than %d \
                \n characters long\nand contain no spaces or *'s\n",
                 g_MaxNameLen );
        do {
                pText = wxGetTextFromUser(msg ,"Enter Class Name",
                         GetClassName());
        } while ((pText!=NULL) && (FALSE==((pText!=NULL)&&(strlen( pText)<32)
                 && (NULL==strstr(pText," ")) && (NULL==strstr(pText,"*")))));
        if (pText != NULL )
                SetClassName( pText );
        //OnPaint();
        Display();
        free( (void*) msg );

}

//#PUBLIC
void CClassWindow::AddParent( CClassWindow* pTemp )
{
        CRelatedList* pNew = new CRelatedList(pTemp);
        pNew->SetNext( m_pAdults);
        m_pAdults = pNew;
}

//#PUBLIC
void CClassWindow::AddChild( CClassWindow* pTemp )
{
        CRelatedList* pNew = new CRelatedList(pTemp);
        pNew->SetNext( m_pChildren);
        m_pChildren  = pNew;
}

//#PUBLIC
void CClassWindow::AddInheritance( void )
{
// moved from OnEvent. called by new parent class which checks for cyclic
//inheritance checks if already parent etc.
        CRelatedList* pTchildren;
        CClassWindow* pChild;

                if (g_pInheritor->IsChild(this) ==TRUE)
                {
                        ((CMainWindow*)((CmwContents*)(this->m_pParent))->
                                GetParent())->SetStatusText(
                                "Sorry, No Cyclic inheritance....aborted");
                        g_pInheritor = NULL;
                } else {
                        // makesure not allready parent
                        pTchildren = m_pChildren;
                        while (pTchildren != NULL)
                        {
                                pChild = pTchildren->GetClass();
                                if (pChild == g_pInheritor)
                                {
                                        ((CMainWindow*)((CmwContents*)(this->
                                                m_pParent))->GetParent())->
                                                SetStatusText(
                                                "Class is allready parent.");
                                        g_pInheritor = NULL;
                                        ((CmwContents*) m_pParent)->SetCursor(
                                                new wxCursor( wxCURSOR_ARROW));
                                        return;
                                }
                                pTchildren = pTchildren->GetNext();
                        }

                        // adding
                        this->AddChild( g_pInheritor);
                        g_pInheritor->AddParent( this );
                        // reset pointer
                        g_pInheritor = NULL;
                        ((CMainWindow*)((CmwContents*)(this->m_pParent))->
                                GetParent())->SetStatusText("");

                        // update graphics
                        ((CmwContents*)this->GetParent())->Clear();
                        ((CmwContents*)this->GetParent())->OnPaint();

                }
                ((CmwContents*) m_pParent)->SetCursor( new
                        wxCursor(wxCURSOR_ARROW));
}

//#PUBLIC
void CClassWindow::RemoveChild( CClassWindow* pTemp )
{
        CRelatedList* pTChild = m_pChildren;
        CClassWindow* pChild =  pTChild->GetClass();
        if ( pChild == pTemp)
        {
                m_pChildren = m_pChildren->GetNext();
                pTChild->SetNext(NULL);
                delete pTChild;
                return;
        } else {
                while (((pTChild->GetNext())!=NULL) &&(((pTChild->GetNext())->
                        GetClass())!=pTemp))
                {
                        pTChild = pTChild->GetNext();
                }
                pTChild->SetNext( ((pTChild->GetNext())->GetNext()));
        }

}

//#PUBLIC
void CClassWindow::RemoveParent( CClassWindow* pTemp )
{
        CRelatedList* pTParent = m_pAdults;
        CClassWindow* pParent =pTParent->GetClass();
        if ( pParent == pTemp)
        {
                m_pAdults = m_pAdults->GetNext();
                pTParent->SetNext(NULL);
                delete pTParent;
                return;
        } else {
                while (((pTParent->GetNext())!=NULL) &&(((pTParent->GetNext())->
                        GetClass())!=pTemp))
                {
                        pTParent = pTParent->GetNext();
                }
                pTParent->SetNext( ((pTParent->GetNext())->GetNext()));
        }
}

//#PUBLIC
void CClassWindow::OnPaint( void )
{
        m_pCNHolder->OnPaint();
        if (m_SeeMembers)
        {
        CMemberFunction* pFun;
        pFun = m_pMemberFuns;
        while (pFun != NULL){
                pFun->OnPaint();
                pFun = pFun->GetNext();
        }
        }
}

//#PUBLIC
Bool CClassWindow::IsChild( CClassWindow* pSearch )
{
        CRelatedList* pTchildren;
        CClassWindow* pChild;
        Bool ret = FALSE;
        pTchildren = m_pChildren;
        while ((pTchildren!=NULL)&& (ret==FALSE))
        {
                pChild = pTchildren->GetClass();
                if (pChild == pSearch )
                        return TRUE;
                ret = pChild->IsChild( pSearch );
                pTchildren = pTchildren->GetNext();
        }
        return ret;
}

//#PUBLIC
void CClassWindow::OnEvent( wxMouseEvent& event)
{
        if (g_pInheritor == NULL)
        {
                CMvPanel::OnEvent( event );
        } else
        if ((g_pInheritor != NULL) && (event.LeftDown() || event.RightDown())
                && (g_pInheritor != this))
        {
                AddInheritance();
        }
}

//#PUBLIC
Bool CClassWindow::SaveMe( void)
{
        FILE *pHeaderFile;
        FILE *pCodeFile;
```

```
char pHeaderName[40];
char pCodeName[40];
sprintf(pHeaderName,"%s.h", GetClassName());
sprintf(pCodeName,"%s.cpp", GetClassName());
wxRemoveFile( pHeaderName );
wxRemoveFile( pCodeName );


// open files
if ( (pHeaderFile = fopen(pHeaderName,"w")) == NULL )
{
        return FALSE;
}
if ( (pCodeFile = fopen(pCodeName,"w")) == NULL)
{
        return FALSE;
}


// 1st write header to both files
fprintf( pHeaderFile, "// \
        \n//This file is generated by Resworb copyright Martin Bramley \
        \n//This file may be edited within reason\n//\n");
fprintf( pCodeFile, "// \
        \n//This file is generated by Resworb copyright Martin Bramley \
        \n//This file may be edited within reason\n//\n");

// 2nd write include files to both files
fprintf( pHeaderFile, "\n//now for include files\n");
fprintf( pCodeFile, "\n//now for include files\n");
CRelatedList* pTAdults;
CClassWindow* pClass;
pTAdults = m_pAdults;

fprintf( pCodeFile, "#include \"%s\"\n",  pHeaderName);
while (pTAdults != NULL)
{
        pClass = pTAdults->GetClass();
        fprintf( pHeaderFile, "#include \"%s.h\"\n", pClass->
                GetClassName() );
        fprintf( pCodeFile, "#include \"%s.h\"\n", pClass->
                GetClassName() );
        pTAdults = pTAdults->GetNext();
}
fprintf( pHeaderFile, "//Finishedincludes\n" );
fprintf( pCodeFile, "//Finishedincludes\n" );
// finished writing in include files


// 3rd write class structure to header file
CClassWindow *pTemp;
pTAdults = m_pAdults;

// write class header and inheritance structure.
if (pTAdults == NULL ) {
        fprintf( pHeaderFile, "class %s {\n", GetClassName() );
} else {
        fprintf( pHeaderFile, "class %s :\n\t", GetClassName() );
        while (pTAdults != NULL)
```

```
		{
			pTemp = pTAdults->GetClass();
			fprintf( pHeaderFile, " public %s", pTemp->
				GetClassName() );
			pTAdults = pTAdults->GetNext();
			if (pTAdults != NULL)
				fprintf( pHeaderFile, ", " );
		}
		fprintf( pHeaderFile, "{\n");
	}
	// finished writing header and inheritaance structure

	//print member variables
	fprintf( pHeaderFile, "//#Start\n%s\n//#Finish\n\n\n", GetData() );

	// function delcalations should go in here!
	// although not just yet!
	{
	CMemberFunction* pFun;
	pFun = m_pMemberFuns;
	while( pFun != NULL )
	{
		char *pFunct;
		pFunct = strdup( pFun->GetData() );
		char *pEndDecl1;
		char *pEndDecl2;
		pEndDecl1 = strstr( pFunct, "{" );
		pEndDecl2 = strstr( pFunct, ":" );

		if ((pEndDecl2 != NULL) && (pEndDecl2 < pEndDecl1)) {
			pEndDecl1 = pEndDecl2;
		}
		pEndDecl1[0] = ';';
		pEndDecl1[1] = '\0';
		//strip out new lines
		do {
			pEndDecl1 = strstr( pFunct, "\n" );
			if (pEndDecl1 != NULL)
				pEndDecl1[0] = ' ';
		} while (pEndDecl1 != NULL);

		// print it to header file
		if (pFun->GetAccess() == PUBLIC) {
			fprintf( pHeaderFile, "%s:\t%s\n","public", pFunct );
		} else if (pFun->GetAccess() == PROTECTED) {
			fprintf( pHeaderFile, "%s:\t%s\n","protected", pFunct );
		} else {
			fprintf( pHeaderFile, "%s:\t%s\n","private", pFunct );
		}
		free( pFunct );
		pFun = pFun->GetNext();
	}
	}
	// finished writing header file
	fprintf( pHeaderFile, "\n}\n");


	// 4th write functions to code file
	CMemberFunction* pFun;
```

```
        pFun = m_pMemberFuns;

        // while there are member functions
        while (pFun != NULL)
        {
                // put start on function flag in file
                //fprintf( pCodeFile, "\n//#Start\n");

                if (pFun->GetAccess()==PUBLIC) {
                        fprintf( pCodeFile, "\n//#PUBLIC\n");
                } else if (pFun->GetAccess()==PROTECTED) {
                        fprintf( pCodeFile, "\n//#PROTECTED\n");
                } else {
                // else private
                        fprintf( pCodeFile, "\n//#PRIVATE\n");
                }

                // copy code to instert classname in function declaration
                char *pCode;
                pCode = strdup( pFun->GetData());

                // Get function name!
                char *pFunName;
                pFunName = pFun->FindFunName();
                // find position of Fun Name;
                char *ppos;
                ppos = strstr( pCode, pFunName );

                // copy rest of fun into tempstr
                char *tempstr;
                tempstr = strdup( ppos );
                // put end of string at ppos
                pCode[ ppos - pCode ] = '\0';
                // append classname and :: to pCode
                strcat( pCode, GetClassName() );
                strcat( pCode, "::" );
                // concat pCode and tempstr
                strcat( pCode, tempstr);
                //write newdata
                fprintf( pCodeFile, pCode);
                // finished writing function code

                // tidy up
                fprintf( pCodeFile, "\n");
                pFun = pFun->GetNext();
                free( pFunName );
                free( pCode);
        }
        fprintf( pCodeFile, "\n//#Finish\n\n\n//Do not delete this line");
        // finished writing function code

        // finished writing data files finish
        fclose( pHeaderFile );
        fclose( pCodeFile );
        return TRUE;
}


//#PUBLIC
```

```
Bool CClassWindow::LoadMe( void)
{
        FILE *pHeaderFile;
        FILE *pCodeFile;
        char pHeaderName[400];
        char pCodeName[400];
        sprintf(pHeaderName,"%s.h", GetClassName());
        sprintf(pCodeName,"%s.cpp", GetClassName());

        // open code files
        if ( (pHeaderFile = fopen(pHeaderName,"r")) == NULL )
        {
                return FALSE;
        }
        if ( (pCodeFile = fopen(pCodeName,"r")) == NULL)
        {
                return FALSE;
        }
        // files are now open

        // lets load the stuff
        // the longest a line may be is 1K (thats huge!!!)
        // thats a loading limit
        char pLine[1024];

        // should rd member variables section of header file here
        //repeat untill //#START
        do {
                GetLine( pHeaderFile, pLine);
        } while (NULL==strstr( pLine, "//#Start"));
        long startofmems, endofmems;

        startofmems = ftell( pHeaderFile );
        do {
                GetLine( pHeaderFile, pLine);
        } while ( NULL==strstr( pLine, "//#Finish"));

        endofmems = ftell( pHeaderFile );
        fseek( pHeaderFile, startofmems, SEEK_SET );
        char* pMemVars;

        // create block to put member variables in
        pMemVars = (char*) malloc( endofmems - startofmems );
        // rd variables
        pMemVars[0] = '\0';
        GetLine( pHeaderFile, pLine);
        do {
                strcat( pMemVars, pLine );
                strcat( pMemVars, "\n");
                GetLine( pHeaderFile, pLine);
        } while (NULL==strstr( pLine, "//#Finish"));
        SetData( pMemVars );
        free( pMemVars );
        // finished loading member variables

        // Now starting to load member functions.
        // skip over #includes and other stuff until 1st funct or eof flag
        // is encountered
        // repeat until //#Start or //#Finish
```

```
do {
	GetLine( pCodeFile, pLine);
} while ( FALSE==FlagFound(pLine));

// continue rding funcs until eof flag is found.
while (0!= strcmp(pLine , "//#Finish"))
{
	// now create item to hold function in
	CMemberFunction* pTemp;
	pTemp = new CMemberFunction( (wxPanel*) this, -1, -1,(int)
		g_WindowWidth, g_SectionHeight);
	pTemp->Move(0,m_currentHeight);
	m_currentHeight = m_currentHeight + g_SectionHeight;
	pTemp->SetNext( m_pMemberFuns );
	m_pMemberFuns = pTemp;
	pTemp->Show( FALSE );

	if (0==strcmp( pLine, "//#PRIVATE")) {
		pTemp->MakeFunc( PRIVATE );
	} else if (0==strcmp( pLine, "//#PUBLIC")) {
		pTemp->MakeFunc( PUBLIC );
	} else if (0==strcmp( pLine, "//#PROTECTED")) {
		pTemp->MakeFunc( PROTECTED );
	} else {
		pTemp->MakeFunc( PUBLIC );
	}
	// find lentgh of function so space may be created
	long startoffun, endoffun;
	startoffun = ftell( pCodeFile );
	do {
		GetLine( pCodeFile, pLine);
	} while ( FALSE==FlagFound(pLine));

	endoffun = ftell( pCodeFile );
	fseek( pCodeFile, startoffun, SEEK_SET );
	char* pFunCode;

	// create block to put function in
	pFunCode = (char*) malloc( endoffun - startoffun );
	// rd func
	pFunCode[0] = '\0';
	GetLine( pCodeFile, pLine);
	do {
		strcat( pFunCode, pLine );
		strcat( pFunCode, "\n");
		GetLine( pCodeFile, pLine);
	} while ( FALSE==FlagFound(pLine));

	// the function is now read in off the disk
	// now strip out the classname:: part!
	char *pPos;
	char* pClname;
	pClname = strdup( GetClassName() );
	strcat( pClname, "::");
	pPos = strstr( pFunCode, pClname );
	free( pClname );
	if (pPos == pFunCode )
	{
		pFunCode[0] = '\0';
		} else {
			pFunCode[ pPos - pFunCode-1] = '\0';
		}
		pPos = pPos + strlen(GetClassName()) + 1;
		pPos[0] = ' ';
		strcat( pFunCode, pPos );

		// put codestream into function
		pTemp->SetData( pFunCode );
		// release the codestream
		free( pFunCode );
		//GetLine( pCodeFile, pLine );
	}
	// finished loading now close the files
	fclose( pHeaderFile );
	fclose( pCodeFile );
	free( pLine );
	return TRUE;
}

//#PUBLIC
int CClassWindow::GetParentList( char* pOutList)
{
	int number=0;
	CRelatedList* pTAdults;
	CClassWindow* pClass;
	pTAdults = m_pAdults;
	//set outlist to be initially empty
	pOutList[0]='\0';
	while (pTAdults!=NULL)
	{
		number++;
		pClass = pTAdults->GetClass();
		sprintf( pOutList, "%s,%s", pOutList, pClass->GetClassName());
		pTAdults = pTAdults->GetNext();
	}
	sprintf( pOutList, "%s,", pOutList);
	return number;
}

//#PUBLIC
int CClassWindow::GetChildList( char* pOutList)
{
	int number=0;
	CRelatedList* pTChildren;
	pTChildren = m_pChildren;
	CClassWindow* pClass;
	//set outlist to be initially empty
	pOutList[0]='\0';
	while (pTChildren!=NULL)
	{
		number++;
		pClass = pTChildren->GetClass();
		sprintf( pOutList, "%s,%s", pOutList, pClass->GetClassName());
		pTChildren = pTChildren->GetNext();
	}
	sprintf( pOutList, "%s,", pOutList);
	return number;
}
```

```
//#PUBLIC
void CClassWindow::EditMembers( void )
{
	CEditWindow* pEdit = new CEditWindow( NULL, this,
		"Edit Member Function",100,100,400,400);
	pEdit->Show(TRUE);
}

//#PUBLIC
void CClassWindow::SetData( const char* pDesc )
{
	if (m_pData != NULL)
		free( m_pData);
	m_pData = strdup( pDesc );
}


//#Finish

// last line
```

8.8 CData.h

```
//******************************** CData.h **************
//
//                  26/11/96
//                  Martin Bramley
//
//*******************************************************

#ifndef DATA_H
#define DATA_H

#ifndef UNIX
#include <wx_prec.h>
#endif

#include <wx.h>

#include "app_def.h"

class CData
{
//#Start
protected:
            char* m_pData;
//#Finish
public:
            CData(void);
//          virtual ~CData( void ) { free( (void*) m_pData ); };
            virtual void SetData( const char*) {cout << " \
                        set data not defined in super-class\n";};
            char* GetData( void );
};

#endif
```

8.9 CData.cpp

```
//**************************** CData.cpp ***************
//
//                    26/11/96
//                    Martin Bramley
//
//*******************************************************

#include <wx_prec.h>
#include <wx.h>
#include <stdlib.h>

#include "CData.h"
#include "assert.h"

//#PUBLIC
CData::CData( void )
{
          m_pData = (char*) malloc(1);
}


//#PUBLIC
char* CData::GetData( void )
{
          return m_pData;
}

//#Finish

// last line
```

8.10 CEditWindow.h

```
//****************************** editwin.h **************
//
//              27/12/96header file for text window
//              Martin Bramley
//
//*****************************************************

#ifndef UNIX
#include <wx_prec.h>
#endif

#include <wx.h>

#ifndef CEDITWIN_H
#define  CEDITWIN_H

#include "CClassWindow.h"
//class CClassWindow;
#include "CData.h"

class CEditWindow: public wxDialogBox
{
//#Start
private:
        wxFrame* m_pFrame;
        void OnSize( int w, int h);
        wxButton* m_pOKbutt;
        wxButton* m_pCANCELbutt;
public:
//      CListPanel* m_pRetClass;
        CData* m_pRetClass;

//#Finish
public:
        CEditWindow(wxFrame*, CData* ,char*,int, int, int, int);
        ~CEditWindow( void );
        wxTextWindow* m_pTextWindow;
        int OnClose( void );
};
#endif
```

8.11 CEditWindow.cpp

```
//****************************** editwin.cpp *************
//
//                      27/12/96code for createing Edit window
//                      Martin Bramley
//
//******************************************************

#include <stdlib.h>
#include <stdio.h>

#include <wx_prec.h>
#include <wx.h>
//#include <malloc.h>

#include "app_def.h"
#include "CEditWindow.h"

#define OK_TAG 1
#define CANCEL_TAG 2

void buttonProc( wxButton& but, wxCommandEvent&)// event )
{

        long tag = (long)but.GetClientData() ;
        if (tag==OK_TAG)
        {
                char* buff;
//              buff = (char*) malloc( g_MaxDescriptionLen );
                CEditWindow* pTemp = (CEditWindow*) but.GetParent();
                buff = (pTemp->m_pTextWindow)->GetContents();
                char *pos;
                pos = strstr( buff, "\r\n");
                while (pos != NULL)
                {
                        buff[pos-buff] = '\0';
                        strcat( buff, pos+1 );
                        pos = strstr( buff, "\r\n");
                }
                if (buff != NULL)
                {
                        (((CEditWindow*)
                                but.GetParent())->m_pRetClass)->SetData( buff );
                        delete buff;
                }
        }
// should delete edit window
        ((CEditWindow*) but.GetParent())->Close();
}


//#PUBLIC
CEditWindow::CEditWindow( wxFrame* parent, CData* pReturnClass,
        char* title,int x,int y,int w,int h) :
        wxDialogBox( parent, title, TRUE,x,y,w,h)
{
```

```
// constructor for user-class window
        m_pTextWindow = new wxTextWindow( this,0,0,w,h-50,
                wxNATIVE_IMPL | wxHSCROLL, "textwindow" );
        m_pTextWindow->SetFont(g_pAppFont);
        m_pTextWindow->DragAcceptFiles(TRUE);
        m_pTextWindow->WriteText( pReturnClass->GetData());
        m_pTextWindow->Show(TRUE);
        // do something so that it resizes properly
        m_pRetClass = pReturnClass;

        m_pOKbutt = new wxButton(this, (wxFunction)&buttonProc, "OK",(w/2)-75,
                h-40,60,30) ;
        m_pOKbutt->SetClientData((char*)OK_TAG) ;

        m_pCANCELbutt = new wxButton(this, (wxFunction)&buttonProc, "Cancel",
                (w/2)+25, h-40,60,30) ;
        m_pCANCELbutt->SetClientData((char*)CANCEL_TAG) ;
}


//#PUBLIC
CEditWindow::~CEditWindow()
{

}


//#PUBLIC
void CEditWindow::OnSize( int w, int h)
{
                //w and h are needed so funct is called properly
        //int width, height;
        //GetClientSize(&width, &height);
        m_pTextWindow->SetSize(0, 0, w, h-50);

        delete m_pOKbutt;
        delete m_pCANCELbutt;

        m_pOKbutt = new wxButton(this, (wxFunction)&buttonProc, "OK",(w/2)-75,
                h-40,60,30) ;
        m_pOKbutt->SetClientData((char*)OK_TAG) ;

        m_pCANCELbutt = new wxButton(this, (wxFunction)&buttonProc, "Cancel",
                (w/2)+25, h-40,60,30) ;
        m_pCANCELbutt->SetClientData((char*)CANCEL_TAG) ;
}
//#PUBLIC
int CEditWindow::OnClose( void )
{
// what happens when a close command is sent to the window
        // hide the window.
        Show( FALSE );
        return TRUE;
}
//#Finish

//last line
```

8.12 CLinkedMenu.h

```
//******************************** linkmenu.h *************
//
//			23/11/96   header for frame containing canvas and
//						menus for class and members
//			Martin Bramley
//
//*******************************************************

#ifndef UNIX
#include <wx_prec.h>
#endif

#include <wx.h>

#ifndef LINKMENU
#define LINKMENU


class CLinkedMenu :  public wxMenu
{
//#Start
private:
		wxPanel* m_pParentFrame;

//#Finish
public:
		CLinkedMenu( wxPanel* );
		~CLinkedMenu();
		void SetContainer( wxPanel* );
		wxPanel* GetContainer( void );
};

void PopupFunction( CLinkedMenu&, wxCommandEvent&);

#endif
```

8.13 CLinkedMenu.cpp

```
//****************************** linkmenu.cpp ************
//
//                  23/11/96   header for frame containing canvas and
//                                         menus for class and members
//                  Martin Bramley
//
//******************************************************

#include<wx_prec.h>
#include<wx.h>

#include "CClassWindow.h"
#include "CLinkedMenu.h"
#include "app_def.h"
#include "CmwContents.h"


//#PUBLIC
CLinkedMenu::CLinkedMenu( wxPanel* parent ) : wxMenu( NULL, (wxFunction)
        PopupFunction)
{
        SetContainer( parent );
}

//#PUBLIC
CLinkedMenu::~CLinkedMenu()
{
        SetContainer(NULL);
}

//#PUBLIC
void CLinkedMenu::SetContainer( wxPanel* pParent)
{
        m_pParentFrame = pParent;
}

//#PUBLIC
wxPanel* CLinkedMenu::GetContainer( void )
{
        return m_pParentFrame;
}


void PopupFunction( CLinkedMenu& menu, wxCommandEvent& event)
{
        switch (event.commandInt )
        {
        case VIEW_MEMBERS:
                ((CClassWindow*)(((CClassNameHolder*)
                        menu.GetContainer())->GetParent()))->SetViewable();
                break;
        case NEW_BASE_CLASS:
                ((CmwContents*)menu.GetContainer())->CreateClass();
                break;
        case ADD_VARIABLE :
```

```
		((CClassWindow*)(((CClassNameHolder*)
			menu.GetContainer())->GetParent()))->EditMembers();
		break;
	case EDIT_NAME :
		((CClassWindow*)(((CClassNameHolder*)
			menu.GetContainer())->GetParent()))->ChangeClassName();
		break;
	case EDIT_DESC :
		((CClassNameHolder*) menu.GetContainer())->EditData();
		break;
	case EDIT_FUNCTION :
		((CMemberFunction*) menu.GetContainer())->EditFunction();
		break;
	case ADD_FUNCTION :
		((CClassWindow*)(((CClassNameHolder*)
			menu.GetContainer())->GetParent()))->AddFunction();
		break;
	case ADD_PARENT:
		((CClassWindow*)(((CClassNameHolder*)
			menu.GetContainer())->GetParent()))->
			ChangeInheritance();
		break;
	case CUT :
		((CClassWindow*)(((CClassNameHolder*)
			menu.GetContainer())->GetParent()))->Cut();
		break;
	case SAVE_CLASS :
		((CClassWindow*)(((CClassNameHolder*)
			menu.GetContainer())->GetParent()))->SaveMe();
		break;
	case DELETE_CLASS :
		{
		CClassWindow* pClass = ((CClassWindow*)(((CClassNameHolder*)
			 menu.GetContainer())->GetParent()));
		CmwContents* pmwC = ((CmwContents* )pClass->GetContainer());
		Bool ret;
		ret = pClass->DeletingClass();
		if (ret==TRUE)
			delete pClass;

		pmwC->Clear();
		pmwC->UpdateLinks();
		}
		break;
	case DELETE_BRANCH :
		{
		CClassWindow* pClass = ((CClassWindow*)(((CClassNameHolder*)
			menu.GetContainer())->GetParent()));
		CmwContents* pmwC = ((CmwContents* )pClass->GetContainer());
		Bool ret;
		ret = pClass->DeletingTree();
		if (ret==TRUE)
			delete pClass;

		pmwC->Clear();
		pmwC->UpdateLinks();
		}
		break;
	case DELETE_FUNCTION :
```

```
                ((CClassWindow*)(((CMemberFunction*)
                        menu.GetContainer())->GetContainer()))->
                        DeleteFunction((CMemberFunction*) menu.GetContainer());
                break;
        case MAKE_PUBLIC :
                ((CMemberFunction*) menu.GetContainer())->MakeFunc(PUBLIC);
                ((CMemberFunction*) menu.GetContainer())->OnPaint();
                break;
        case MAKE_PROTECTED :
                ((CMemberFunction*) menu.GetContainer())->MakeFunc(PROTECTED);
                ((CMemberFunction*) menu.GetContainer())->OnPaint();
                break;
        case MAKE_PRIVATE :
                ((CMemberFunction*) menu.GetContainer())->MakeFunc(PRIVATE);
                ((CMemberFunction*) menu.GetContainer())->OnPaint();
                break;
        default :
                cout << "error \anot defined yet!\n";
                break;
        }
}

//#Finish
```

8.14 CListPanel.h

```
//******************************* listpnl.h *************
//
//			26/11/96
//			Martin Bramley
//
//*******************************************************

#ifndef LISTPNL_H
#define LISTPNL_H

#ifndef UNIX
#include <wx_prec.h>
#endif

#include <wx.h>

#include "CData.h"
#include "app_def.h"

class CListPanel :  public wxPanel, public CData
{
//#Start
protected:
		CListPanel* m_pNext;
private:
		int m_ID;
public:
		wxPanel* m_pParent;
//#Finish
public:
		CListPanel( wxPanel*);
		CListPanel( wxPanel*,int,int,int,int);
		~CListPanel();
		void GetPosition( int*, int*);
		void SetNext( CListPanel* );
		CListPanel* GetNext( void );
		int GetID(void);
		wxPanel* GetContainer(void);
};

#endif
```

8.15 CListPanel.cpp

```
//**************************** listpnl.cpp ***************
//
//              26/11/96
//              Martin Bramley
//
//*******************************************************

#include <wx_prec.h>
#include <wx.h>
#include <stdlib.h>

#include "CData.h"
#include "CListPanel.h"
#include "assert.h"

int maxFrameNo=0;

//#PUBLIC
CListPanel::CListPanel( wxPanel* pParent) :
                wxPanel( (wxWindow*) pParent,-1,-1,-1,-1, wxBORDER), CData()
{

        m_pNext = NULL;
        m_pParent = pParent;
        maxFrameNo = maxFrameNo + 1;
        m_ID = maxFrameNo;
        wxDC* ptemp = GetPanelDC();
        ptemp->SetFont( g_pAppFont );
//      m_pData = (char*) malloc( 1 );
}

//#PUBLIC
CListPanel::CListPanel( wxPanel* pParent,int x,int y,int w,int h) :
                wxPanel( (wxWindow*) pParent,x,y,w,h, wxBORDER), CData()
{

        m_pNext = NULL;
        m_pParent = pParent;
        //maxFrameNo = maxFrameNo + 1;
        //m_ID = maxFrameNo;
        wxDC* ptemp = GetPanelDC();
        ptemp->SetFont( g_pAppFont );
//      m_pData = (char*) malloc( 1 );
}


//#PUBLIC
CListPanel::~CListPanel()
{

//      free( (void*) m_pData);
}

//#PUBLIC
wxPanel* CListPanel::GetContainer( void)
```

```
{
	return m_pParent;
}

//#PUBLIC
void CListPanel::SetNext( CListPanel* pNext)
{
	m_pNext = pNext;
}

//#PUBLIC
CListPanel* CListPanel::GetNext( void )
{
	return m_pNext;
}

//#PUBLIC
int CListPanel::GetID()
{

	return m_ID;
}

//#PUBLIC
void CListPanel::GetPosition(int *x, int *y)
{

	wxPanel::GetPosition( x,y);
}

//#PUBLIC
//char* CListPanel::GetData( void )
//{
//	return m_pData;
//}

//#Finish

// last line
```

8.16 CMainWindow.h

```
//****************************** mw.h ***************
//
//                  21/12/96header file for main window
//                  Martin Bramley
//
//******************************************************

#ifndef UNIX
#include <wx_prec.h>
#endif

#include <wx.h>

#ifndef CMAINWIN_H
#define  CMAINWIN_H

#include "CmwContents.h"
#include "CMenuBar.h"
#include "CListPanel.h"
#include "CData.h"

//Finishedincludes
class CMainWindow :
        public wxFrame, public CData
{
//#Start
private:
        CMenuBar*       m_pMenubar;
        CmwContents*    m_pContents;
//#Finish
public:
        Bool GetClassName( FILE*, char* );
        CMainWindow(wxFrame*, char*, int, int, int, int, long, char*);
        void OnMenuCommand(int id);
        ~CMainWindow(void);
        void OnPaint( void );
        void OnSize( int, int);
        void ImportClass( void );
        int OnClose( void );
        void SaveCurrent( void );
        void LoadProject( int );
        void LoadthisProject( char* );
        Bool DiscardEdits( void );
        void EditPrefs( void );
        void SetData( const char* );
        void ShowHelp( void );
};
#endif
```

8.17 CMainWindow.cpp

```
//****************************** mw.cpp *************
//
//          21/12/96code for createing main window
//          Martin Bramley
//
//*****************************************************

#include <wx_prec.h>
#include <wx.h>
#include <stdlib.h>

#include "app_def.h"
#include "CMainWindow.h"
#include "CmwContents.h"
#include "CMenuBar.h"
#include "CClassWindow.h"
#include "CEditWindow.h"
//Finishedincludes

extern CClassWindow* g_pInheritor;

//#PUBLIC
Bool CMainWindow::GetClassName( FILE* pInFile, char* pOutString )
{
        int i,c;
        for (i=0; ((c=fgetc( pInFile))!=',') && !feof( pInFile); i++)
                pOutString[i] = char(c);

        if (feof(pInFile))
        {
                fclose( pInFile);
                cout << "unexpected end of file\n";
                return FALSE;
        }
        pOutString[i] = '\0';
        return TRUE;
}

//#PUBLIC
CMainWindow::CMainWindow( wxFrame* parent, char* title, int posx, int posy,
                 int height, int width, long style, char *name) :
                wxFrame( parent, title, posx, posy, height, width, style,
                 name), CData()
{
        m_pContents = new CmwContents(this);
        m_pContents->OnPaint();
        m_pMenubar = new CMenuBar();
        SetMenuBar( m_pMenubar );
        CreateStatusLine(1);
        CClassWindow* pClassListhd;
        CClassWindow* pClass;

                        pClassListhd = m_pContents->GetContents();
                        while (pClassListhd != NULL) {
                                pClass = pClassListhd;
```

```
                                        pClassListhd = (CClassWindow*)
                                                  (pClassListhd->GetNext());
                                        delete pClass;
                              }
                              m_pContents->SetContents( NULL );
                              m_pContents->Clear();
}

//#PUBLIC
CMainWindow::~CMainWindow()
{
          delete m_pContents;
          delete m_pMenubar;
}

//#PUBLIC
void CMainWindow::OnSize( int x, int y)
{
          wxFrame::OnSize( x, y);
          m_pContents->OnPaint();

}

//#PUBLIC
int CMainWindow::OnClose( void )
{
// what happens when a close command is sent to the window
          // hide the window.
          wxFrame::Show( FALSE );
          delete this;
          return TRUE;
}

//#PUBLIC
void CMainWindow::OnPaint( void )
{
//        wxFrame::OnPaint();
//        m_pContents->OnPaint();
}

//#PUBLIC
void CMainWindow::ImportClass( void )
{
          char *pTemp;
          char *pDir;
          char *pFileName;
          char *pClassName;
          int pos;

          CClassWindow* pClass;

          pTemp = wxFileSelector("Import a class",NULL,NULL,NULL,"*.h");
          if (pTemp){
                    pDir = strdup( wxPathOnly( pTemp ) );
                    pFileName = strdup( wxFileNameFromPath( pTemp ) );

                    pClassName = (char*) malloc( 2000 );

                    pos = strstr( pFileName, ".h" ) - pFileName;
```

```
                strncpy( pClassName, pFileName, pos);
                pClassName[pos] = '\0';

                wxSetWorkingDirectory( wxPathOnly( pTemp ) );

                pClass = m_pContents->GetClass( pClassName );
                if (pClass==NULL)
                {
                        pClass = new CClassWindow(m_pContents);
                        pClass->SetClassName( pClassName );
                        // add new class to list of maintained classes.
                        pClass->SetNext( m_pContents->GetContents() );
                        m_pContents->SetContents( pClass );
                        // Load data
                        pClass->LoadMe();
                        // set attributes of new class
                        pClass->Display( 0, 0);
                } else {
                        wxMessageBox("Class already exists","Importing Error",
                                wxOK|wxCENTRE);

                }
                free( pDir);
                free( pTemp );
                free( pFileName );
                free( pClassName );
        }
}


//#PUBLIC
void CMainWindow::LoadthisProject( char *pTemp )
{
        CClassWindow* pClass;
        CClassWindow* pRelation=NULL;
        FILE *pDataFile;
        char *pFileName;
        char pClassName[32];
        int posx, posy;
        Bool RetVal = TRUE;
        char *pDir;

        pDir = strdup( wxPathOnly( pTemp ) );
        pFileName = strdup( wxFileNameFromPath( pTemp ) );
        wxSetWorkingDirectory( wxPathOnly( pTemp ) );

        if ( (pDataFile = fopen(pFileName,"r")) == NULL )
        {
                wxMessageBox("Loading Error", "Aborting Load", wxOK,
                        (wxFrame*)this);
                return;
        }

        while( !feof( pDataFile ) && RetVal)
        {
                RetVal = GetClassName( pDataFile, pClassName );
                fscanf( pDataFile, "%d,%d,",&posx, &posy);
                // create new class
                pClass = m_pContents->GetClass( pClassName );
```

```
				if (pClass==NULL)
				{
					pClass = new CClassWindow(m_pContents);
					pClass->SetClassName( pClassName );
					// add new class to list of maintained classes.
					pClass->SetNext( m_pContents->GetContents() );
					m_pContents->SetContents( pClass );
				}
				pClass->Show( FALSE );
				// Load data
				pClass->LoadMe();
				// set attributes of new class
				pClass->Display( posx, posy);
				// rd Child Line
				int num,counter;
				fscanf( pDataFile, "%d,", &num);
				for (counter=0;counter<num; counter++)
				{
					GetClassName( pDataFile, pClassName );

					pRelation = m_pContents->GetClass( pClassName );
					if (pRelation == NULL)
					{
						pRelation = new CClassWindow( m_pContents );
						pRelation->SetClassName( pClassName );
						pRelation->SetNext( m_pContents->GetContents());
						m_pContents->SetContents( pRelation );
					}
					//link to it
					g_pInheritor = pRelation;
					pClass->AddInheritance();
					g_pInheritor = NULL;
				}
				fscanf(pDataFile, "\n");
				// finished reading
				if (RetVal == FALSE )
				{
					wxMessageBox("Loading Error", "Aborting Load", wxOK,
						(wxFrame*)this);
				}
			}
//		m_pContents->UpdateLinks();
		free( pDir );
		free( pFileName );
		fclose( pDataFile );
		pClass = (CClassWindow*) (m_pContents->m_pContents);
		while (pClass != NULL)
		{
			pClass->Show(TRUE);
			pClass = (CClassWindow*) pClass->GetNext();
		}
}

//#PUBLIC
void CMainWindow::LoadProject( int flag )
{
		char *pTemp;
		if (flag==0) {
			if (FALSE==DiscardEdits()) return;
```

```
        } else {
                if (wxOK!= wxMessageBox("Problems may a rise with duplicate file names",
                        "Import Project", wxOK | wxCANCEL,(wxFrame*)this))
                                return;
        }
        pTemp = wxFileSelector("Load a project",NULL,NULL,NULL,"*.rwb");
        if (pTemp)
                LoadthisProject( pTemp );
}


//#PUBLIC
void CMainWindow::SaveCurrent( void )
{
        CClassWindow* pClass;
        CClassWindow* pClassListhd;
        FILE *pDataFile;
        char pFileName[256];
        char *pTemp;
        int posx, posy;
        Bool RetVal = TRUE;
        pTemp = wxFileSelector("Save a project",NULL,NULL,NULL,"*.rwb");
        if (pTemp)
        {
                wxSetWorkingDirectory( wxPathOnly( pTemp ) );
                sprintf(pFileName,"%s",wxFileNameFromPath( pTemp));
                int namelen;
                namelen = strlen( pFileName);
                if (0!=strcmp( &pFileName[namelen-4], ".rwb"))
                        sprintf(pFileName, "%s.rwb", pFileName );

                if ( (pDataFile = fopen(pFileName,"w")) == NULL )
                {
                        wxMessageBox("Saving Error", "Killing Resworb?", wxOK,
                                (wxFrame*)this);
                        OnClose();
                }

                pClassListhd = m_pContents->GetContents();
                while ((pClassListhd != NULL) && (RetVal)) {
                        pClass = pClassListhd;
                        pClassListhd = (CClassWindow*)(pClassListhd->GetNext());
                        RetVal = pClass->SaveMe();
// write class name
                        fprintf(  pDataFile, "%s,", pClass->GetClassName() );
// write position
                        pClass->GetPosition( &posx, &posy);
                        fprintf( pDataFile, "%d,%d,", posx,posy);
//write children
                        // getnumber of Children and string
                        char* pList;
                        pList = (char*) malloc(1024*40);
                        int num = pClass->GetChildList(pList);
                        fprintf( pDataFile, "%d%s\n", num, pList);
                        free( pList);
// end of datafile saving.
                        if (RetVal == FALSE)
                        {
                                wxMessageBox("Saving Error", "Killing Resworb?",
                                        wxOK,(wxFrame*)this);
                                OnClose();
                        }
                }
                fclose( pDataFile );
        }
}

//#PUBLIC
Bool CMainWindow::DiscardEdits( void )
{
        CClassWindow* pClassListhd;
        CClassWindow* pClass;
        if (NULL == m_pContents->GetContents())
        {
                m_pContents->Clear();
                m_pContents->SetContents( NULL );
                return TRUE;
        }
        if (wxOK == wxMessageBox("discard current project", "New Project",
                wxOK | wxCANCEL,(wxFrame*)this) )
        {
                pClassListhd = m_pContents->GetContents();
                while (pClassListhd != NULL)
                {
                        pClass = pClassListhd;
                        pClassListhd = (CClassWindow*)(pClassListhd->GetNext());
                        delete pClass;
                }
                m_pContents->Clear();
                m_pContents->SetContents( NULL );
                return TRUE;
        }
        return FALSE;
}
//#PUBLIC
void CMainWindow::EditPrefs( void )
{
        FILE* pDefaultsFile;
        char pName[1000];
        strcpy(pName ,getenv("HOME"));
        strcat(pName , "/.resworb.def");
        if ( (pDefaultsFile = fopen(pName,"r")) != NULL ) {
                long startoffile, endoffile;
                startoffile = ftell( pDefaultsFile );
                fseek( pDefaultsFile, 0, SEEK_END );
                endoffile = ftell( pDefaultsFile );
                fseek( pDefaultsFile, 0, SEEK_SET );
                free( (void*) m_pData );
                m_pData = (char*) malloc( endoffile- startoffile + 2);
                fread( m_pData, (endoffile - startoffile), 1, pDefaultsFile);
                fclose( pDefaultsFile );
        } else {
                m_pData = strdup( " \
// an appdef file for resworb ln 2=bgcol 3=Public 4=private 5=protected 6=fontsize 7=font \
\nBLUE\nRED\nGREEN\nBLUE\n12\nDEFAULT\n\n");
        }

        CEditWindow* pEdit = new CEditWindow( NULL, this,
                "Edit Class Description",100,100,400,400);
        pEdit->Show(TRUE);
}

//#PUBLIC
void CMainWindow::SetData( const char *pData)
{
        if (NULL == strstr(pData, "RESWORB HELP") )
        {
        if (m_pData != NULL)
                free( m_pData);
        m_pData = strdup( pData );

        FILE* pDefaultsFile;
        char pName[1000];
        strcpy(pName ,getenv("HOME"));
        strcat(pName , "/.resworb.def");
        if ( (pDefaultsFile = fopen(pName,"w")) != NULL ) {
                fwrite(m_pData, strlen(m_pData), 1, pDefaultsFile);
                fclose( pDefaultsFile );
        }
        } else {
                //do nothing
        }

}

//#PUBLIC
void CMainWindow::ShowHelp( void )
{
        free( (void*) m_pData );
        m_pData = strdup( "\t\tRESWORB HELP\n\tDO NOT CHANGE THE ABOVE LINE \
            \n Loading and Saving Projects\n\tjust use the file menus or the \
                \n\tassociated short cuts\n\nUsing the mouse \
                \n\tLeft Button :- Drag Class\n\tMiddle Button :- Does Nothing \
                \n\tRight Button :- Context Menu\n");

        CEditWindow* pEdit = new CEditWindow( NULL, this,
                "Edit Class Description",50,50,500,500);
        pEdit->Show(TRUE);
}

//#PUBLIC
void CMainWindow::OnMenuCommand(int id)
{
        switch (id)
        {
        case EDIT_PREFS:
                EditPrefs();
                break;
        case LOAD_PROJECT :
                LoadProject(0);
                break;
        case SAVE_PROJECT :
                SaveCurrent();
                break;
        case NEW_PROJECT :
                DiscardEdits();
                break;
```

```
	case IMPORT_CLASS :
		ImportClass();
		break;
	case IMPORT_PROJECT :
		LoadProject(1);
		break;
	case QUIT_RESWORB :
		if (wxOK == wxMessageBox("Quit without Saving!",
			"Quit Resworb?", wxOK | wxCANCEL,(wxFrame*)this) )
			OnClose();
		break;
	case ABOUT_RESWORB:
		wxMessageBox("Resworb\n A Graphical class Browser AND generator \
			\n By Martin Bramley Feb 1997","About Resworb",
			 wxOK|wxCENTRE);
		break;
	case SHOW_HELP:
		ShowHelp();
		break;
	default :
		break;
	}
}
//#Finish


// a few extra lines!
```

8.18 CMemberFunction.h

```
//**************************** memfun.h *****************
//
//                10/02/97   code for managing member functs
//                                    and display
//                Martin Bramley
//*****************************************************

#ifndef UNIX
#include<wx_prec.h>
#endif

#include<wx.h>

#include "app_def.h"

#ifndef CMEMBERFUNCT_H
#define CMEMBERFUNCT_H

#include "CMenuPanel.h"
#include "CClassWindow.h"

class CMemberFunction: public CMenuPanel
{
//#Start
private:
        CMemberFunction* m_pNext;
        wxPanel* m_pParent;
        char* m_pFunctName;
        char* m_pCode;
        int m_Access;
//#Finish
public:
        CMemberFunction( wxPanel*,int,int,int,int);
        ~CMemberFunction( void );
        char* FindFunName( void );
        wxPanel* GetContainer( void );
        void EditFunction(void);
        void OnPaint( void );
        CMemberFunction* GetNext();
        void SetNext( CMemberFunction* );
        void GetDims( float&, float& );
        char* GetFunctionName( void );
        void SetData( const char* );
        void MakeFunc( int );
        int GetAccess(void);
};

#endif
```

8.19 CMemberFunction.cpp

```
//***************************** clname.cpp ***************
//
//                10/02/97   code for managing member functs
//                                and display
//                Martin Bramley
//*****************************************************
#include <stdlib.h>
#include <stdio.h>

#include <string.h>
#include <wx_prec.h>
#include <wx.h>

#include "CClassWindow.h"
#include "CMemberFunction.h"
#include "CEditWindow.h"

const char* pConstFunct= "void NoName( void )\n{ \
        \n//Enter Code as usual except comments must \
        \n be with in the function or after it. Not before the declaration \
        \n}\n";

void MakeFunctionMenu( CLinkedMenu* pMenu )
{
        pMenu->Append( EDIT_FUNCTION, "Edit Function");
        pMenu->Append( DELETE_FUNCTION, "Delete Function");
        pMenu->Append( MAKE_PRIVATE, "Make Private");
        pMenu->Append( MAKE_PROTECTED, "Make Protected");
        pMenu->Append( MAKE_PUBLIC, "Make Public");
}

//#PUBLIC
CMemberFunction::CMemberFunction( wxPanel *parent, int x=-1, int y=-1,
        int w=-1, int h=-1) :  CMenuPanel( parent, x,y,w,h)
{
//for some really obscure reason the following 2 lines must occur before
// the following malloc!
        m_pParent = parent;
        MakeFunctionMenu( m_pMenu );

        m_pNext = NULL;
        SetData( pConstFunct);
        m_pFunctName = FindFunName();
        m_Access = PUBLIC;
}

//#PUBLIC
char* CMemberFunction::FindFunName( void )
{
        char* pData;
        char* pCode;
        char *pCrap;
        char pName[1024];

        pCode = strdup( GetData() );
        pData = pCode;

        pCrap=strstr( pData, "(");
        pData[ pCrap - pData] = '\0';
        // if virtual in fun decl jump over!
        pCrap=strstr( pData, "virtual");
        if (pCrap != NULL)
        {
                pData = pCrap + strlen("virtual");
        }
        while (pData[0]==' ')
                pData++;
        // should be at start of return type name
        pCrap = strstr( pData, " ");
        if (pCrap != NULL)
                pData = pCrap+1;
        // now stepped over return type
        while (pData[0]==' ')
                pData++;
        if (pData[0]=='*')
                pData++;
        while (pData[0]==' ')
                pData++;
        sscanf( pData, "%s", pName );
        delete pCode;
        return strdup( pName );
}

//#PUBLIC
CMemberFunction::~CMemberFunction( void )
{
        free( m_pFunctName);
        if (m_pNext != NULL)
                delete m_pNext;
}

//#PUBLIC
void CMemberFunction::GetDims( float& w, float& h)
{
        char* funName = GetFunctionName();
        wxDC* pDC = GetPanelDC();
        pDC->GetTextExtent( funName, &w, &h);
}

//#PUBLIC
void CMemberFunction::EditFunction( void )
{
        CEditWindow* pEdit = new CEditWindow( NULL, this,
                "Edit Member Function",100,100,400,400);
        pEdit->Show(TRUE);
}

//#PUBLIC
char* CMemberFunction::GetFunctionName( void )
{
        return m_pFunctName;
}

//#PUBLIC
```

```
void CMemberFunction::OnPaint( void )
{
if (((CClassWindow*) m_pParent)->m_SeeMembers)
        {
        wxDC* pDC = GetPanelDC();
        pDC->Clear();
        int w=0,h=0;
        GetSize( &w, &h);
        pDC->DrawRectangle((float) 0, (float) 0,(float) w, (float) h);

        wxColour *OldCol;
        OldCol =  (wxColour*) malloc(sizeof(wxColour));
        *OldCol = pDC->GetTextForeground();

        if (m_Access == PROTECTED){
                pDC->SetTextForeground( g_pColProtected );
        } else if (m_Access == PRIVATE){
                pDC->SetTextForeground( g_pColPrivate );
        } else if (m_Access == PUBLIC){
                pDC->SetTextForeground( g_pColPublic );
        }

        pDC->DrawText( GetFunctionName(), (float) 5, (float)
                ((g_SectionHeight /2) - (g_SectionHeight /4)));

        pDC->SetTextForeground( OldCol );
        free( OldCol );
        }

}

//#PUBLIC
CMemberFunction* CMemberFunction::GetNext( void )
{
        return m_pNext;
}

//#PUBLIC
wxPanel* CMemberFunction::GetContainer( void )
{
        return m_pParent;
}

//#PUBLIC
void CMemberFunction::SetNext( CMemberFunction* pNext )
{
        m_pNext = pNext;
}

//#PUBLIC
void CMemberFunction::SetData( const char* pDesc )
{
        if (m_pData != NULL)
                free( m_pData);
        m_pData = strdup( pDesc );
        m_pFunctName= FindFunName();
        ((CClassWindow*)m_pParent)->Display();
}
```

```
//#PUBLIC
void CMemberFunction::MakeFunc( int access )
{
	m_Access = access;
}

//#PUBLIC
int CMemberFunction::GetAccess( void )
{
	return m_Access;
}

//#Finish

// last line
```

8.20 CMenuBar.h

```
//****************************** menu.h ***************
//
//              21/12/96   header file for main window menus
//              Martin Bramley
//
//*****************************************************

#ifndef UNIX
#include <wx_prec.h>
#endif

#include <wx.h>

#ifndef MENU_H
#define  MENU_H

class CMenuBar: public wxMenuBar
{
//#Start
// no mem vars!
//#Finish
public:
        CMenuBar(void);
        ~CMenuBar(void);
};
#endif
```

8.21 CMenuBar.cpp

```
//****************************** menu.cpp ***************
//
//              21/12/96code for main window menus
//              Martin Bramley
//
//*****************************************************

#include <wx_prec.h>
#include <wx.h>

#include "app_def.h"
#include "CMenuBar.h"

//#PUBLIC
CMenuBar::CMenuBar(void)
{
        wxMenu *file_menu = new wxMenu;
        wxMenu *pref_menu = new wxMenu;
        wxMenu *help_menu = new wxMenu;

        file_menu->Append(NEW_PROJECT, "&New\tCtrl+N",
                "Create new resworb project");

        file_menu->Append(LOAD_PROJECT, "&Open Project\tCtrl+O",
                "Opens a Resworb Project");

        file_menu->Append(SAVE_PROJECT, "&Save Project\tCtrl+S",
                "Saves current Project");

        file_menu->AppendSeparator();

        file_menu->Append(IMPORT_CLASS, "&Import Class\tCtrl+I",
                "Quits Resworb with no saving");

        file_menu->Append(IMPORT_PROJECT, "&Import Project",
                "Import other project");

        file_menu->AppendSeparator();

        file_menu->Append(QUIT_RESWORB, "&Quit Resworb\tCtrl+X",
                "Quits Resworb with no saving");


        pref_menu->Append(EDIT_PREFS, "Edit Preferences",
                "Opens preferences file");

        help_menu->Append( ABOUT_RESWORB, "&About","About Resworb");

        help_menu->Append( SHOW_HELP, "&Help", "Display Help Window");

        this->Append( file_menu, "&File");
        this->Append( pref_menu, "&Prefences");
        this->Append( help_menu, "&Help");
}
```

```
//#PUBLIC
CMenuBar::~CMenuBar(void)
{

}

//#Finish

// the extra line
```

8.22 CMenuPanel.h

```
//****************************** menupnl.h *************
//
//              26/11/96
//              Martin Bramley
//
//*****************************************************

#ifndef MENUPNL_H
#define MENUPNL_H

#ifndef UNIX
#include <wx_prec.h>
#endif

#include <wx.h>

#include "CLinkedMenu.h"
#include "CListPanel.h"

class CMenuPanel :  public CListPanel
{
//#Start
protected:
        CLinkedMenu* m_pMenu;
//#Finish
public:
        CMenuPanel( wxPanel*, int, int, int, int);
        ~CMenuPanel();
        void OnEvent( wxMouseEvent& event );
        virtual void OnPaint( void ) {};
        void SetNext( CMenuPanel* );
        CMenuPanel* GetNext( void );
};

#endif
```

8.23 CMenuPanel.cpp

```
//**************************** menupnl.cpp ***************
//
//			26/11/96
//			Martin Bramley
//
//******************************************************
#include <stdlib.h>
#include <wx_prec.h>
#include <wx.h>

#include "CMvPanel.h"
#include "CmwContents.h"
#include "CMainWindow.h"
#include "CMenuPanel.h"
#include "assert.h"
#include "app_def.h"
#include "CEditWindow.h"
#include "CClassWindow.h"

extern CClassWindow* g_pInheritor;
extern CMvPanel* g_pMover;



//#PUBLIC
CMenuPanel::CMenuPanel( wxPanel* pParent, int x, int y, int w, int h) :
	CListPanel( pParent,x,y,w,h)
{

	m_pMenu = new CLinkedMenu( this);
//	SetTextBackground(g_pColWHITE);
}

//#PUBLIC
CMenuPanel::~CMenuPanel()
{

	delete m_pMenu;
}

//#PUBLIC
void CMenuPanel::SetNext( CMenuPanel* pNext)
{
	CListPanel::SetNext( (CListPanel*) pNext);
}

//#PUBLIC
CMenuPanel* CMenuPanel::GetNext( void )
{
	return ( (CMenuPanel*) (CListPanel::GetNext()));
}


//#PUBLIC
void CMenuPanel::OnEvent( wxMouseEvent& event )
```

```
{
	float x,y;
	event.Position(&x, &y);
	if (event.RightDown() && (g_pInheritor ==NULL) && (g_pMover == NULL)) {
		PopupMenu( m_pMenu, x, y);
	} else {
		int tempx =(int) x;
		int tempy = (int) y;
		ClientToScreen( &tempx, &tempy );
		((CMvPanel*) m_pParent)->ScreenToClient( &tempx, &tempy);
		event.x = (float) tempx;
		event.y = (float) tempy;
		((CMvPanel*) m_pParent)->OnEvent( event );
	}
}



//#Finish

// last line
```

8.24 CMvPanel.h

```
//******************************* movepnl.h **************
//
//              23/11/96   header for frame containing canvas and menus
//                                          for class and members
//              Martin Bramley
//
//********************************************************

#ifndef MOVEPNL_H
#define MOVEPNL_H

#ifndef UNIX
#include <wx_prec.h>
#endif

#include <wx.h>

#include "CListPanel.h"

class CMvPanel :  public CListPanel
{
//#Start
private:
        int m_XOffSet;
        int m_YOffSet;
//#Finish
public:
        CMvPanel( wxPanel*);
        ~CMvPanel();
        void OnEvent( wxMouseEvent& event );
        void Move( int, int);
        virtual void OnPaint( void ) {};
        void SetNext( CMvPanel* );
        CMvPanel* GetNext( void );
};

#endif
```

8.25 CMvPanel.cpp

```
//**************************** movepnl.cpp ***************
//
//                  23/11/96   code for frame containing canvas and menus
//                                             for class and members
//                  Martin Bramley
//
//*******************************************************

#include <string.h>
#include <wx_prec.h>
#include <wx.h>

#include "CClassWindow.h"
#include "CMvPanel.h"
#include "CmwContents.h"
#include "CMainWindow.h"
#include "assert.h"
#include "app_def.h"

CMvPanel* g_pMover=NULL;
float dx=float(-1);
float dy=float(-1);

//#PUBLIC
CMvPanel::CMvPanel( wxPanel* pParent) :
                  CListPanel( pParent)
{
          // a couple of hacks to deal with application bars
          m_XOffSet = 0;
          m_YOffSet = 0;
}

//#PUBLIC
CMvPanel::~CMvPanel()
{ }

//#PUBLIC
void CMvPanel::SetNext( CMvPanel* pNext)
{
          CListPanel::SetNext( (CListPanel*) pNext);
}

//#PUBLIC
CMvPanel* CMvPanel::GetNext( void )
{
          return ( (CMvPanel*) (CListPanel::GetNext()));
}

//#PUBLIC
void CMvPanel::Move(int x, int y)
{
#if defined(wx_msw) && !defined(__WATCOMC__)
          wxWindow::Move(x,y);
#else
// hopefully allows -ve positions by overriding defualt Move code!
// this may get very nasty
          Widget drawingArea = (Widget) handle;
          Bool managed = XtIsManaged(borderWidget ? borderWidget :scrolledWindow);

          if (managed)
                  XtUnmanageChild (borderWidget ? borderWidget : scrolledWindow);
          XtVaSetValues(drawingArea, XmNresizePolicy, XmRESIZE_ANY, NULL);

          XtVaSetValues (borderWidget ? borderWidget : scrolledWindow,
                  XmNx, x, XmNy, y, NULL);

          if (managed)
                  XtManageChild (borderWidget ? borderWidget : scrolledWindow);
          XtVaSetValues(drawingArea, XmNresizePolicy, XmRESIZE_NONE, NULL);
#endif
}

//#PUBLIC
void CMvPanel::OnEvent( wxMouseEvent& event )
{
          static float x,y;
          event.Position(&x, &y);
          if (event.LeftDown()) {
                  g_pMover = this;

                  int tx, ty;
                  tx = (int)x; ty = (int)y;
                  ClientToScreen( &tx, &ty );
                  m_pParent->ScreenToClient( &tx, &ty);

                  ((CmwContents*)m_pParent)->SetCursor( new
                          wxCursor(wxCURSOR_CROSS));

                  int x2, y2, h, w;
                  GetPosition( &x2, &y2);
                  GetSize( &w, &h);
                  ( (CmwContents *)m_pParent)->SetRect( x2,y2,w,h);
                  ( (CmwContents *)m_pParent)->ShowRect( TRUE );
                  dx = float(tx - x2);
                  dy = float(ty - y2);
          ((CMainWindow*)( ( (CmwContents*)
                  m_pParent)->GetParent()))->SetStatusText("Moving Class ...");
          } else {
                  int tx =(int) x;
                  int ty = (int) y;
                  ClientToScreen( &tx, &ty );
                  m_pParent->ScreenToClient( &tx, &ty);
                  event.x = (float) tx;
                  event.y = (float) ty;
                  m_pParent->OnEvent( event );
          }
}

//#Finish
// last line
```

8.26 CmwContents.h

```
//****************************** mwcont.h ***************
//
//                 24/12/96header file for stuff in main window
//                 Martin Bramley
//
//*****************************************************

#ifndef  MWCONT_H
#define MWCONT_H

#ifndef UNIX
#include <wx_prec.h>
#endif

#include <wx.h>
#include "CClassWindow.h"
#include "CLinkedMenu.h"

class CmwContents: public wxPanel
{
//#Start
private:
        wxPen* m_pBluePen;
        wxPen* m_pBlackPen;
        wxPen* m_pSolidBlackPen;
        CLinkedMenu* m_pMenu;
        int m_rectx, m_recty, m_rectw, m_recth;
        Bool m_ShowRect;
        Bool m_positnew;
public:
        CClassWindow* m_pContents;
//#Finish
public:
        CmwContents(wxFrame*);
        CClassWindow* GetContents(void);
        CClassWindow* GetClass( char* );
        void SetContents( CClassWindow* );
        virtual ~CmwContents(void) ;
        CClassWindow* GetFrames( void );
        void OnPaint(void);
        void OnEvent(wxMouseEvent& event);
        void ShowRect( Bool );
        void SetRect( int, int, int, int);
        void OnScroll( wxCommandEvent& );
        Bool AddingNew();
        void SetBluePen(void);
        void SetSolidBlackPen(void);
        void SetBlackPen(void);
        void UpdateLinks( void );
        void RemoveClass( CClassWindow* );
        void CreateClass();

};
#endif
```

8.27 CmwContents.cpp

```
//****************************** mwcont.cpp *************
//
//              24/12/96code stuff in main window
//              Martin Bramley
//
//******************************************************

#include <stdlib.h>
#include <stdio.h>

#include <wx_prec.h>
#include <wx.h>
#include <malloc.h>

#include "assert.h"
#include "app_def.h"
#include "CmwContents.h"
#include "CClassNameHolder.h"
#include "CMainWindow.h"

extern CMvPanel* g_pMover;
extern float dx;
extern float dy;

//#PUBLIC
CmwContents::CmwContents(wxFrame *frame): wxPanel(frame)
{
        m_rectx =0; m_recty=0; m_rectw=0; m_recth=0;
        m_ShowRect = FALSE;

        wxBrush* pBrush = new wxBrush(*g_pColBGround, wxSOLID);
        SetBackground( pBrush );

        m_pBluePen = new wxPen(*g_pColBGround,1, int(wxSOLID));
        m_pBlackPen = new wxPen(*g_pColBLACK,1, int(wxSHORT_DASH));
        m_pSolidBlackPen = new wxPen(*g_pColBLACK,1, int(wxSOLID));

        // determine width of windows to contain variable names
        // maximum no of chars
        g_MaxNameLen = g_MaxNameLen +2;

        // buffer to contain chars to find width
        char* buff;
        buff = (char*) malloc( g_MaxNameLen);
        memset(buff, 'A', g_MaxNameLen);

        // add end of string terminator
        buff[g_MaxNameLen] = '\0';

        // Get DC and find width
        wxDC* pDC = GetPanelDC();
        float TempMaxHeight;

        SetFont( g_pAppFont );
        pDC->SetFont( g_pAppFont);
```

```
	float to_be_converted;
	g_WindowWidth=0;
	pDC->GetTextExtent( buff, &to_be_converted, &TempMaxHeight);
	g_WindowWidth = (int) to_be_converted;

	g_WindowWidth += 5; g_SectionHeight = (int) (TempMaxHeight * 2);

	// now free the buffer memory
	free( (void*) buff );

	m_pContents = NULL;

	SetScrollbars(20,20,50,50,4,4);
	SetScrollRange( wxHORIZONTAL, 100);
	SetScrollRange( wxVERTICAL, 100);
	EnableScrolling( TRUE, TRUE);
	m_positnew=FALSE;

	m_pMenu = new CLinkedMenu( this );
	m_pMenu->Append( NEW_BASE_CLASS, "New Class");
	OnPaint();
}

//#PUBLIC
CmwContents::~CmwContents(void)
{
	CClassWindow *temp;

	while (m_pContents != NULL)
	{
		temp = (CClassWindow*) (m_pContents->GetNext());
		m_pContents->SetNext( NULL );
		delete m_pContents;
		m_pContents = temp;
	}


}

//#PUBLIC
CClassWindow* CmwContents::GetContents( void )
{
	return m_pContents;
}

//#PUBLIC
void CmwContents::SetContents( CClassWindow* pClass)
{
	m_pContents = pClass;
}

//#PUBLIC
void CmwContents::UpdateLinks( void )
{
	CClassWindow* pClass = m_pContents;
	int x,y;
	int w,h;
	while (pClass != NULL)
	{
        pClass->GetPosition( &x, &y);
        pClass->GetSize( &w, &h);
        pClass->UpdateChildLinks(x,y,float(w),float(h));
        pClass = ((CClassWindow*)pClass->GetNext());
    }
}

//#PUBLIC
void CmwContents::OnScroll( wxCommandEvent& event )
{
    CClassWindow* pTemp;
    int xmov=-1, ymov=-1;
    int x,y;
    GetVirtualSize(&x, &y);
    xmov = x / GetScrollRange( wxHORIZONTAL );
    ymov = y / GetScrollRange( wxVERTICAL );

    static int prevx=0, prevy=0;
    int XOffset = 0;
    int YOffset = 0;

    if (event.extraLong == wxHORIZONTAL)
    {
        XOffset=xmov * (event.commandInt - prevx);
        prevx = event.commandInt;
    } else {
        YOffset=ymov * (event.commandInt - prevy);
        prevy = event.commandInt;
    }


    pTemp = m_pContents;
    int curx, cury;
    while (pTemp != NULL) {
        pTemp->GetPosition( &curx, &cury);
        pTemp->Move( curx-XOffset, cury-YOffset );
        pTemp = (CClassWindow*) pTemp->GetNext();
    }
    Clear();
    OnPaint();
}


//#PUBLIC
void CmwContents::OnPaint(void)
{
// Define the repainting behaviour
    if (m_ShowRect) {
        //Clear();
        IntDrawLine(m_rectx, m_recty, m_rectx+m_rectw, m_recty);
        IntDrawLine(m_rectx+m_rectw, m_recty, m_rectx+m_rectw,
            m_recty+m_recth);
        IntDrawLine(m_rectx+m_rectw, m_recty+m_recth, m_rectx,
            m_recty+m_recth);
        IntDrawLine(m_rectx, m_recty+m_recth, m_rectx, m_recty);
    }
    UpdateLinks();
}
```

```
//#PUBLIC
CClassWindow* CmwContents::GetFrames( void )
{
        return m_pContents;
}

//#PUBLIC
void CmwContents::OnEvent(wxMouseEvent& event)
{
        float x, y;
        event.Position(&x, &y);

        if ((g_pMover != NULL) && event.LeftIsDown() && event.Dragging() &&
                                                        (m_positnew == FALSE)) {
                int w,h;
                GetSize( &w,&h);
                // remove old outline
                SetBluePen();
                OnPaint();
                // finished remove outline

                m_rectx = (int) (x - dx);
                m_recty = (int) (y - dy);

                if (m_rectx < -1) {
                        m_rectx = 0;
                }
                if (m_recty < -1) {
                        m_recty = 0;
                }
                if (((m_rectx + m_rectw) > w)){
                        m_rectx = w-m_rectw-12;
                }
                if (((m_recty + m_recth) > h) ){
                        m_recty = h-m_recth -12;
                }
                SetBlackPen();
                OnPaint();
        } else
        if (event.LeftUp() && (g_pMover != NULL) &&( m_positnew ==FALSE)) {
                SetBluePen();
                ((CClassWindow*)g_pMover)->Display( m_rectx, m_recty,
                        float(m_rectw),float(m_recth));
                g_pMover->OnPaint();
                ShowRect( FALSE );
                g_pMover = NULL;
                OnPaint();
                ((CMainWindow*)GetParent())->SetStatusText("");
                SetCursor( new wxCursor( wxCURSOR_ARROW) );
        } else
        if (event.RightDown())
        {
                dx = x;
                dy = y;
                PopupMenu( m_pMenu, x,y);
        }

}
```

```
//#PUBLIC
Bool CmwContents::AddingNew( void )
{
	return m_positnew;
}


//#PUBLIC
void CmwContents::SetBluePen( void )
{
	SetPen( m_pBluePen );
}

//#PUBLIC
void CmwContents::SetBlackPen( void )
{
	SetPen( m_pBlackPen);
}

//#PUBLIC
void CmwContents::SetSolidBlackPen( void )
{
	SetPen( m_pSolidBlackPen);
}


//#PUBLIC
void CmwContents::SetRect( int x, int y, int w, int h)
{
	m_rectx=x;
	m_recty=y;
	m_rectw=w;
	m_recth=h;
}

//#PUBLIC
void CmwContents::ShowRect( Bool flag )
{
	m_ShowRect = flag;
}

//#PUBLIC
void CmwContents::RemoveClass( CClassWindow* pClass)
{
	CClassWindow* pTemp;
	pTemp = m_pContents;
	if (pTemp == pClass)
	{
		m_pContents = (CClassWindow*)(m_pContents->GetNext());
		return;
	} else {

		while ((((CClassWindow*)pTemp->GetNext()) != pClass) &&
			(((CClassWindow*)pTemp->GetNext()) != NULL)) {
			pTemp = (CClassWindow*) pTemp->GetNext();
		}
	}
```

```
        assert( pTemp->GetNext() != NULL );
        pTemp->SetNext( (CClassWindow*)
                (((CClassWindow*)pTemp->GetNext())->GetNext()));
}

//#PUBLIC
void CmwContents::CreateClass()
{

        // create new class and give it a name
        CClassWindow* pTemp = new CClassWindow( this );

        // insert new class into list managed by frame!
        pTemp->SetNext( m_pContents);
        m_pContents = pTemp ;
//      pTemp->SetFocus();
        pTemp->Display( int(dx), int(dy));
        dx =float( 0);
        dy =float( 0);

        int xxx,yyy;
        CMainWindow* pt = (CMainWindow*) GetParent();
        pt->GetSize( &xxx, &yyy);
#ifndef wx_msw
        pt->SetSize(xxx,yyy);
#else
        int nx, ny;
        pt->GetPosition(&nx,&ny);
        pt->SetSize( nx,ny,xxx,yyy);
#endif
        OnPaint();
}

//#PUBLIC
CClassWindow* CmwContents::GetClass( char* pClNameIn)
{
        CClassWindow* pClass;
        Bool found = FALSE;
        pClass = m_pContents;
        while ((pClass != NULL) && (found==FALSE))
        {
                if (0==strcmp(pClNameIn, pClass->GetClassName()))
                {
                        found = TRUE;
                } else  {
                        pClass = (CClassWindow*) pClass->GetNext();
                }
        }
        return pClass;
}

//#Finish
```

8.28 CRelatedList.h

```
//**************************** classlist.h ******************

#ifndef UNIX
#include <wx_prec.h>
#endif

#include <wx.h>


#ifndef CLASSLIST_H
#define CLASSLIST_H

class CClassWindow;

class CRelatedList
{
//#Start
private:
	CRelatedList* m_pNext;
	CClassWindow* m_pClass;
//#Finish
public:
	CRelatedList(CClassWindow*);
	~CRelatedList(void);
	CRelatedList* GetNext(void);
	void SetNext (CRelatedList*);
	CClassWindow* GetClass(void);
	void SetClass (CClassWindow*);
};
#endif
```

8.29 CRelatedList.cpp

```
//**************************** classlst.cpp *****************

#include <wx_prec.h>
#include <wx.h>


#include "CClassWindow.h"
#include "CRelatedList.h"

//#PUBLIC
CRelatedList::CRelatedList(CClassWindow* pClass)
{
	SetNext(NULL);
	SetClass(pClass);
}

//#PUBLIC
CRelatedList::~CRelatedList(void)
{
	if (m_pNext != NULL)
		delete m_pNext;
}

//#PUBLIC
CRelatedList* CRelatedList::GetNext(void)
{
	return m_pNext;
}

//#PUBLIC
void CRelatedList::SetNext (CRelatedList* pNext)
{
	m_pNext = pNext;
}


//#PUBLIC
CClassWindow* CRelatedList::GetClass(void)
{
	return m_pClass;
}

//#PUBLIC
void CRelatedList::SetClass (CClassWindow* pClass)
{
	m_pClass = pClass;
}

//#Finish

// last line
```

8.30 CResworbApp.h

```
//*************************** App.H *********************
//
//                    18/11/96Resworb Main application header file
//                    Martin Bramley
//
//*****************************************************

// This file defines the class structure of the application.

#ifdef WX_PRECOMP
#include <wx_prec.h>
#endif

#include <wx.h>
#include "app_def.h"
//#Finishedincludes

class CResworbApp: public wxApp
{
//#Start
private:
            // Application has one window.
public:
            CMainWindow* m_pAppWindow;
//#Finish
public:
            CResworbApp(void) ;
            ~CResworbApp() ;
            wxFrame* OnInit( void );
};
```

8.31 CResworbApp.cpp

```
//************************ App.CPP **********************
//
//                  18/11/96Resworb Main application file
//                  Martin Bramley
//
//*****************************************************


#include <wx_prec.h>
#include <wx.h>
#include <stdio.h>

#include "CMainWindow.h"
#include "CResworbApp.h"


// The instance of the application
CResworbApp     ResworbApp;
// Global application font
wxFont* g_pAppFont = NULL;

// Global application colours
wxColour* g_pColBLACK=NULL;
wxColour* g_pColWHITE=NULL;
wxColour* g_pColBGround=NULL;
wxColour* g_pColProtected=NULL;
wxColour* g_pColPrivate=NULL;
wxColour* g_pColPublic=NULL;


// max name length of variables
int g_MaxNameLen = 32;
int g_MaxDescriptionLen = 512;

// worked out from (max name length* 2) +2 (for * and space) + 2 for ()
// this is actaully set in the creation of the main window! mwcont.cpp
int g_WindowWidth;
int g_SectionHeight;

extern void GetLine( FILE* , char* );

//#PUBLIC
CResworbApp::CResworbApp( void )
{
        // Application constructor
        // There should only be one instance of the application
        // which is made statically at run time hence beware
        // what come in here!
}

//#PUBLIC
CResworbApp::~CResworbApp()
{
}
```

```
//#PUBLIC
wxFrame* CResworbApp::OnInit( void )
{
        g_pColBLACK = wxTheColourDatabase->FindColour("BLACK");
        g_pColWHITE = wxTheColourDatabase->FindColour("WHITE");

        FILE* pDefaultsFile;
        char pName[1000];
        strcpy(pName ,getenv("HOME"));
        strcat(pName , "/.resworb.def");
        if ( (pDefaultsFile = fopen(pName,"r")) != NULL )
        {
                char pLine[100];
                // header line
                GetLine( pDefaultsFile, pLine );
                // bg colour
                GetLine( pDefaultsFile, pLine );
                g_pColBGround = wxTheColourDatabase->FindColour(pLine);

                // pub col
                GetLine( pDefaultsFile, pLine );
                g_pColPublic = wxTheColourDatabase->FindColour(pLine);

                // pri col
                GetLine( pDefaultsFile, pLine );
                g_pColPrivate = wxTheColourDatabase->FindColour(pLine);

                // pro col
                GetLine( pDefaultsFile, pLine );
                g_pColProtected = wxTheColourDatabase->FindColour(pLine);

                // font specs
                int fsize;
                GetLine( pDefaultsFile, pLine);
                sscanf( pLine, "%d\n", &fsize );
                GetLine( pDefaultsFile, pLine);

                if (strstr( pLine, "DEFAULT" ) != NULL){
                        g_pAppFont = new wxFont(fsize, wxDEFAULT, wxNORMAL,
                                wxBOLD );
                }else if (strstr( pLine, "DECORATIVE" ) != NULL ) {
                        g_pAppFont = new wxFont(fsize, wxDECORATIVE, wxNORMAL,
                                wxBOLD );
                }else if (strstr( pLine, "ROMAN" ) != NULL) {
                        g_pAppFont = new wxFont(fsize, wxROMAN, wxNORMAL,
                                wxBOLD );
                } else if (strstr( pLine, "SCRIPT" ) != NULL) {
                        g_pAppFont = new wxFont(fsize, wxSCRIPT, wxNORMAL,
                                wxBOLD );
                } else if (strstr( pLine, "SWISS" ) != NULL) {
                        g_pAppFont = new wxFont(fsize, wxSWISS, wxNORMAL,
                                wxBOLD );
                } else {
                        g_pAppFont = new wxFont(fsize, wxMODERN, wxNORMAL,
                                wxBOLD );
                }

                // finihsed
                fclose( pDefaultsFile );
```

```
	} else {
		cout << "no .resworb.def file found! using defaults\n";
		g_pColBGround = wxTheColourDatabase->FindColour("YELLOW");
		g_pColPublic = wxTheColourDatabase->FindColour("RED");
		g_pColPrivate = wxTheColourDatabase->FindColour("BLUE");
		g_pColProtected = wxTheColourDatabase->FindColour("GREEN");
		g_pAppFont = new wxFont(10, wxMODERN, wxNORMAL, wxBOLD );
	}

	// create Application window
	m_pAppWindow = new CMainWindow(NULL, "resworB::Browser",
					0,0,600,600, wxDEFAULT_FRAME,"resworb");

	// display created window
	m_pAppWindow->Show(TRUE);

	m_pAppWindow->wxWindow::SetSize(600,600);
	if (argc == 2)
		m_pAppWindow->LoadthisProject( argv[1]);

	m_pAppWindow->OnPaint();

	// return application window so that it may be managed
	return m_pAppWindow;
}

//#Finish


// a few lines
```

8.32 Makefile

```
# Program name.
PROGRAM = resworb

# Which compiler you want.
CXX = g++
CC = gcc

# Uncomment this line for debugging support.
DEBUG = -g -Wall
# Uncomment this line for normal optimisation.
#DEBUG = -O2

# These lines set up includes and libraries. You will probably need to
# change them.
INCLUDE = -I../wx/include/base -I../wx/include/x
LIBRARIES = -L/user.sh/common/lib\
	-L../wx/lib \
	../wx/lib/libwx_motif.a -lXm -lXt -lX11 -lcurses -lm



# You shouldn't need to change anything beyond here...

CFLAGS = $(DEBUG) -Dwx_motif -DDEBUG='0'

OBJS =	assert.o \
	CClassNameHolder.o \
	CListPanel.o \
	CMenuPanel.o \
	CmwContents.o \
	CClassWindow.o \
	CMainWindow.o \
	CMvPanel.o \
	CEditWindow.o \
	CMemberFunction.o \
	CRelatedList.o \
	CLinkedMenu.o \
	CMenuBar.o \
	CResworbApp.o \
	CData.o




all: $(PROGRAM)

shared: $(OBJS)
	$(CXX) -o $(PROGRAM) $(OBJS) \
		/user.sh/common/lib/libwx_motif.sa \
		-L/user.sh/common/lib -L../lib \
		-lXmu \
		-lwx_motif -lXm -lXt -lX11 -lcurses -lm

$(PROGRAM): $(OBJS)
	$(CXX) -o $(PROGRAM) $(OBJS) $(LIBRARIES)
```

```
clean: purge
	$(RM) -f *.o $(PROGRAM)

purge:
	$(RM) -f *%

depend:
	$(CC) -M $(INCLUDE) $(CFLAGS) \
		`echo $(OBJS) | sed -e "s/\.o/.cpp/g"` > depends.mk

include depends.mk

%.o: %.cpp
	$(CXX) $(CFLAGS) $(INCLUDE) -c $<
```